Structural Damage Prognosis of Elastico-Mechanoluminescent Composites via Artificial Neural Networks

Undergraduate Honors Thesis

Presented in Partial Fulfillment of the Requirements for the Undergraduate Research Distinction in the College of Engineering Education of The Ohio State University

By

Ahmed Salah Mohamed Kassem Mohamed

The Ohio State University

2020

Thesis Committee

Advisor: Prof. Vishnu Sundaresan

Prof. Manoj Srinivasan

Abstract

In a world foreseeing the commerciality of autonomous vehicles in the forthcoming years, there is an imperative need for reliable structural damage prognosis (SDP) to actively monitor the health of the structural components of the vehicles. In this thesis, an Artificial Neural Network (ANN)-based SDP algorithm is constructed by correlating a mechanoluminescent composite coupon's luminescence data to health of the coupon. In this work an underlying relationship between mechanoluminescent light intensity and structural age is developed in the frequency domain by performing the Short-time Fourier Transform (STFT) on the measured light intensity data. The obtained STFT spectrogram, is a function of frequency and structural age, and encodes vital information regarding structural health of the coupon in the form of time-varying amplitudes at various constitutive frequencies, transforming the SDP problem to a classification problem solvable by an ANN. The ANN algorithm was trained and tested yielding high accuracies in predicting age in real-time for the training phase, yet poor results for the testing phase due to inter-sample variation and, hence, possible decision-boundary overfitting. Development of an ANN algorithm for predicting health of the composite structures in addition to provision of condition-based maintenance instead of planned maintenance will reduce unnecessary machine downtime as well as avoid unforeseen catastrophic failures.

Acknowledgments

I would love to acknowledge my sincere gratitude to my advisor Prof. Vishnu Sundaresan for his guidance, his mentorship, and for being a continuous source of knowledge throughout this research project. The lessons gained from this experience, both professionally and academically, are going to be carried with me forever.

I would also wish to thank Sujasha Gupta for her constant help and directions in utilizing the software programs and the literature reviews related to this project.

Finally, I would like to thank all my lab colleagues for their support, insightful comments and simulating.

Table of Contents

List of Figures

Chapter 1. Introduction

In an autonomous vehicle, due to the limited human-vehicle interaction, changes or anomalies undergone by any of the system's components, particularly those related to propagating damage, are less detectable compared to these encountered by a manually driven vehicle. Additionally, many of the applications entailing autonomous vehicles, among which are extraterrestrial exploration, aeronautics, military defense, excavation, etc., operate under challenging physical and loading conditions, necessitating the presence of a real-time structural deterioration monitoring tool embedded in the system that alerts the passenger of befalling damage.

Damage, in this study, is defined as an intentional or unintentional change experienced by any of the system's constituent components that may detrimentally affect the system's current or future performance. For damage detection, numerous methods were introduced over the past decades, as summarized by Doebling [1]. The state-of-the-art of these methods can be grouped into the context of one or multiple of these jointly interconnected domains [2]: Condition Monitoring (CM), Structural Health Monitoring (SHM), Structural Damage Prognosis (SDP), Statistical Process Control (SPC), and Nondestructive Testing (NDT) – also widely known as Nondestructive Evaluation or Nondestructive Inspection (NDT or NDI). These overlapping damage detection domains are categorized into either *local-based* methods or *global-based* methods [3]. A *local-*

based damage detection technique involves the direct assessment of structures at a component or subcomponent length-scales. Moreover, *local-based* techniques require knowledge of potential, yet accessible, damage locations *a-priori* to optimize the inspection process. Furthermore, *local-based* technologies demand qualified personnel to operate the inspection procedure, which raises their operating cost. Conventional NDT methods, such as Visual inspection [4], Ultrasonic and Sonic NDT [5],[6] , Shearography [7], Electromagnetic and Eddy Current NDT [8], Radiographic NDT [9], and Optical and Non-optical Thermology [10], constituting the majority of *local-based* inspection methods, are considered off-line damage identification processes, or process executed during a machine's downtime.

Unlike a *local-based* damage detection method, a *global-based* one is capable of assessing the condition of an entire structure by conducting vibration measurements at one location, where the damage vicinity may be dissimilar to the measurement location. *Global-based* damage detection quantifies damage by observing changes in local modal properties (e.g. local stiffness and damping) that are reflected on global vibration characteristics (e.g. natural frequencies and mode shapes). Due to its reliance on vibration-based approaches, *Global-based* technologies requisite the availability of installed sensors in the monitoring structure to record temporal responses, which are later integrated with numerical methods to evaluate structural degradation. However, for complex structures and systems subject to highly varying loading scenarios, a global approach for damage detection becomes more taxing [1], for the accuracy of the extrapolated global vibration characteristics is proportional to the nodal density or number of installed sensors. In

addition, conventional sensors are often bulky, thus unsuitable for applications involving miniature structural components [3]. To overcome the caveats associated with increased nodal densities and sensors' largeness, this thesis proposes the use of Elastico-Mechanoluminescent (EML) composites, polydimethylsiloxane (PDMS) matrix with zinc sulfide mechanoluminescent phosphor particles as reinforcement, where each particle acts as a sensor *per se*–as a low-cost alternative to conventional transducers for *global-based* damage detection.

Mechanoluminescence (ML) is a physical phenomenon in which a material emits cold light, as opposed to thermal radiation, when mechanically stimulated. ZNS phosphors are commercial-grade particles that manifest intense ML during elastic deformations; therefore, they are inherently pertinent for applications demanding structural damage prognosis. Structural damage prognosis (SDP) is a progressive *global-based* damage identification process that encompasses estimating a structure's residual life in real-time, unlike structural health monitoring, which is restricted to damage characterization and assessment. The integration of EMLs in the implementation of SDP relies on ML particles being self-recovering crystals. Self-recovering crystals, the likes of copper-doped and manganese-doped Zinc sulfide phosphors, are renewable luminescence sources that require neither any pre-irradiation nor any post-irradiation for EML recovery [11]. Consequently, self-recovery ML particles are deemed reliable for applications that involve repeatability, such as cyclic loading, which is vital for the implementation of SDP.

In literature, there has been a few efforts in building models that correlate the outputted light intensity with the applied stress, strain, or any derived quantities [12]. This correlation

was deemed promising in laying the fundamentals of analytically approaching SDP; however, the main drawback to such efforts is the high sensitivity of the models to material-related properties. That is, to successfully implement such models for new materials, knowledge of a number of quantum-level material properties is needed beforehand, which is very unpragmatic and high-priced. This shifts the attention to data-driven approaches, with a particular focus on machine learning being a well-fit domain for large datasets, which is usually the case with mechanical testing data of high sampling rates. In an attempt to incorporate supervised machine learning, multivariate regression, to fit a model that can relate the output luminescence, stress, and strain to the rate of change in the material Young's modulus, Gupta *et al.* [13] was successful in constructing the model only for the linear portion of the time-luminescence curve. However, to predict the change in material's elastic stiffness based on the nonlinear part of the curve, more correlations related to parameters like stress rate, strain rate, power density, and even more parameters is needed, which are all unknown up to date. The complexity of developing models, both empirical and analytical, that can relate the light intensity to material deterioration or any SDP interpreting parameter, forces the use of predictive data-based approaches, particularly Neural Networks.

Neural Networks are a sector of supervised machine learning that mimics the human brain in its parallel computational style and is believed to share the learning theory that our brains use to process patterns in acquired data and stimuli. For SDP, there has been attempts to calculate a component's residual life by implementing Neural Networks, yet none of which were implemented on smart particulate composites [14]-[16]. Additionally, none of these

achieved SDP on smart materials using the methodology set in this study. Compared to the majority of literature, which addresses the underlying theory of neural networks in a brief manner and, instead, provides references to journals and books for the reader to obtain further information, this document targets readers with limited or no background in Artificial Neural Networks and signal processing methods. In addition, this thesis is intended to serve as a guidebook in literature, perhaps, leading to higher appreciation of such tools.

In this thesis, we report a novel ANN based SDP algorithm for elastico-mechanoluminescent (EML) composites, where important structural information is extracted from modal analysis, particularly STFT of emitted light intensity and fed as inputs to the ANN architecture to classify the health of the coupon based on underlying structural damage to the coupon. EMLs elastomeric composite coupons impregnated with ZnS phosphors are given a sinusoidal load until fatigue failure is undergone, and the emitted light intensity and strain from the coupon in this period is recorded. Collected raw luminescence data, is segmented into various structural ages based on their percentage of failure as observed in the fatigue curve displayed in Figure 2(a). The STFT analysis is performed to obtain time-varying amplitude spectrums of light intensity at each windowed age, making them functions of both time and frequency. Peak values of amplitude in the STFT 3-D spectrum shown in Figure 3(a), which correspond to the brightest spots in the spectrogram displayed in Figure 3(b), are selected as input features for the ANN algorithm to identify the underlying pattern of emitted light intensity at the different ages.

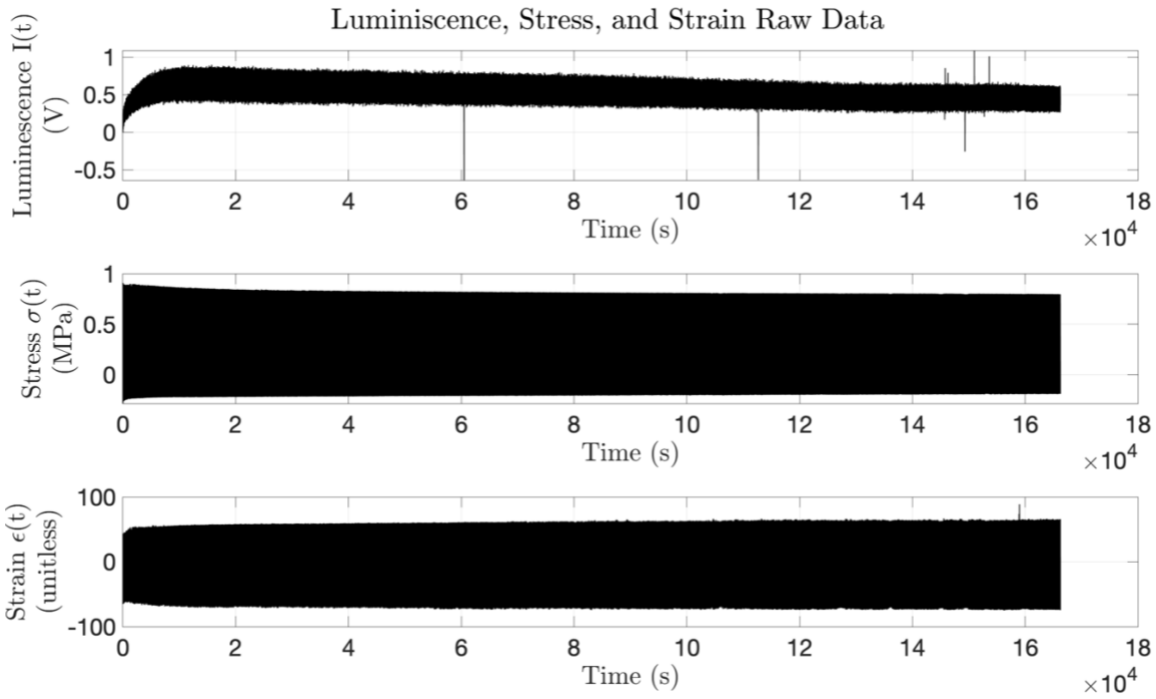


Figure 1: Raw Luminescence, Stress, and Strain data as functions of time. These measurements are for one coupon from start to failure.

The NN was then trained using 50 training examples from five sample coupons (each coupon has 100 structural ages based on percentage of failure) to enable the construction of a weights matrix, whose construction pattern is determined based on the underlying light intensity pattern in frequency domain at a given health state. Training of the algorithm was achieved by implementation of feed-forward propagation and backpropagation algorithm. Optimization of the SDP algorithm was achieved by advanced optimization function predefined in MATLAB to minimize a continuous differentiable multivariate function. Finally, the algorithm was tested to predict the health state of the sixth coupon utilizing the constructed weights matrices.

Chapter 2. Discrete Short-time Fourier Transform (STFT)

2.1 Introduction: Data Acquisition of Luminescence

To implement SDP, data acquisition of the luminescence is essential. Accordingly, to obtain the luminescence data needed for SDP, six coupons EML composite coupons were longitudinally loaded at a frequency of 20 Hz up until mechanical failure of the coupon (refer to Krishnan *et al.* [16] and its supporting information for coupon fabrication and testing methods). During testing, one input parameter: stress, in addition to two output parameters: strain and EML intensity, are recorded from start to failure over millions of actuation cycles. Raw data of the three parameters are plotted in Figure 1. Figure 2(b) shows five cycles of EML intensity captured at three different time intervals: I) early age, II) intermediate age, and III) near-failure age. Note that the frequency of the EML intensity is twice the 20 Hz loading frequency as light is emitted during both the sample's tension half-cycle and compression half-cycle. In case I, the wavelet topology is comprised of superimposed prominent high frequency and low frequency components. Contrary to case I, the wavelet shape is majorly constructed by low frequency components in case II. Further, in case III, the waveform displays both high and low frequency components, with high frequency components being more dominant in comparison to the waveform in case I. This dynamicity in the amplitude spectrum can be observed in Figure 2(c).

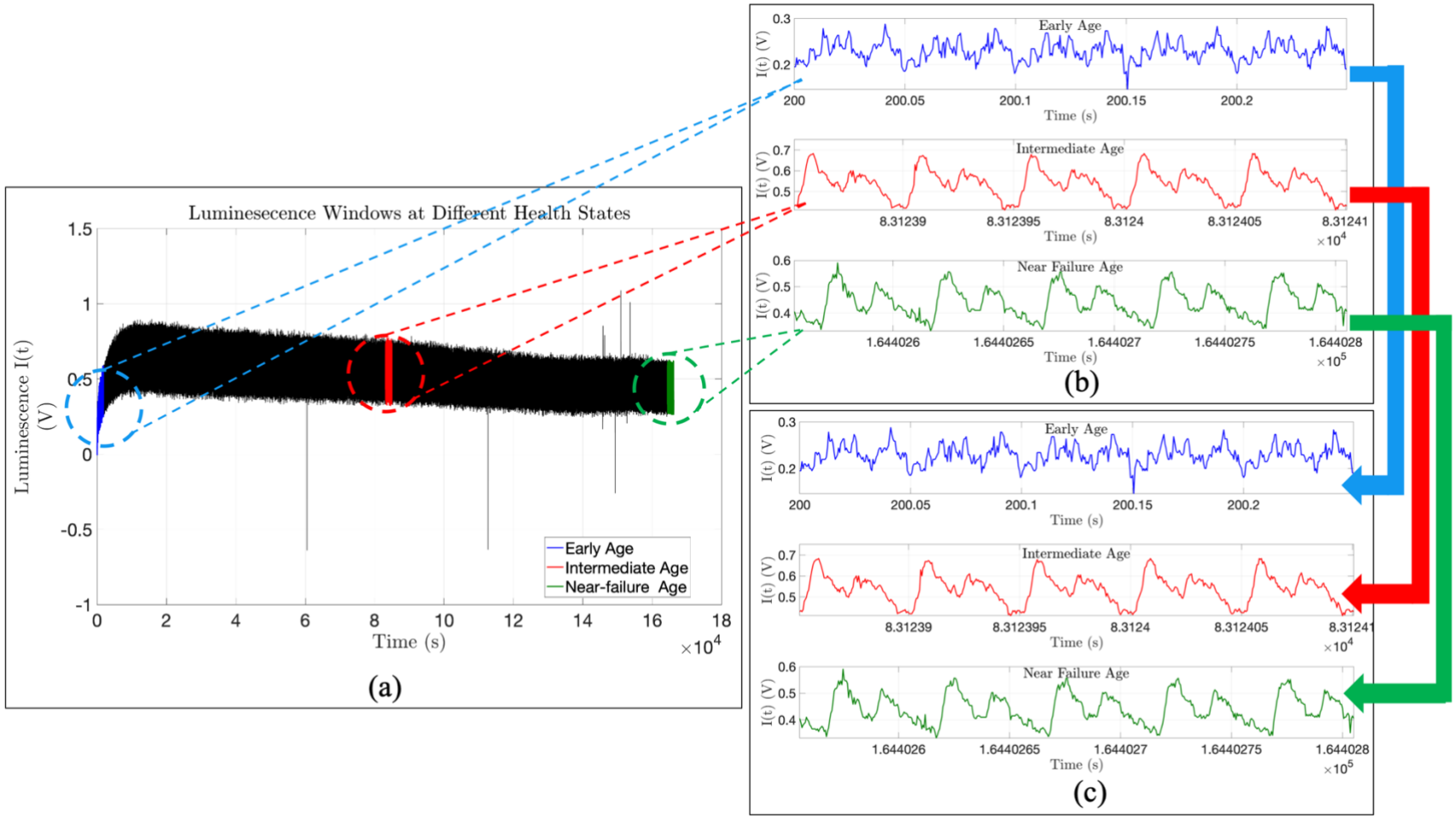


Figure 2(a) Luminescence raw data with three windows of three distinct ages highlighted in different colors. Figure 2(b): Five cycles of the three selected windows: i) luminescence at an early age, ii) luminescence at an intermediate age, and iii) luminescence at a near-failure age. Figure 2(c): The frequency domain representation of the five cycles of luminescence obtained using DTFT

In all three cases and throughout the entire loading period, the input load's amplitude, mean, and frequency are kept unchanged. However, as concluded from cases I, II, and III, the response of the composite's system underwent change due to alteration in the system's mechanical properties resulting from continuous mechanical loading. As reported by Krishnan *et al.* [16], due to continuous loading, the coupons' interfacial binding, which coheres the EML particles to the PDMS matrix, weakens gradually, ensuing less interfacial particle-matrix contact. Consequently, the stress transfer efficiency from the matrix the EML particles is significantly reduced, resulting in less EML intensity stimulation. To analyze the system's time-varying EML intensity signal, a decomposition of the discrete signal's magnitude-varying sinusoidal frequency components (discrete STFT), also referred to as the Windowed Discrete Fourier Transform in other literature, was implemented.

2.2 Discrete-time Fourier Transform

To introduce how STFT was performed in this work, first, the Discrete-time Fourier Transform (DTFT) is presented, for STFT is fundamentally based on the DTFT. Discrete-time Fourier Transform (DTFT) is an integral transform that inputs a periodic discrete-time signal and outputs the constituting amplitudes of the signal as a function of frequency. For a finite time-dependent signal, the frequency domain representation can be obtained by performing the DTFT. Implementing DTFT is given by this group of equations [17]:

$$Y[f_k] = \frac{2}{N}\sum_{r=0}^{N-1} y[r]e^{-\frac{j2\pi rk}{N}}, \qquad r = 0, 1, 2, \dots, (N-1) \tag{1}$$

$$f_k = k\delta f, \qquad k = 0, 1, 2, \dots, \left(\frac{N}{2} - 1\right) \tag{2}$$

$$\delta f = \frac{1}{N\delta t} \tag{3}$$

where, N is the temporal signal's total number of sample points, δt is the equally spaced temporal signal's resolution, $y[r]$ (the square brackets are to denote a discrete function) is the temporal signal's amplitude at the index r – can also be expressed in terms of the time $r\delta t$ –, and j is the imaginary number equivalent of $\sqrt{-1}$. For any point in the DTFT signal with an amplitude of $Y[f_k]$, f_k is the index of that DTFT point, and δf is the signal's equally spaced frequency resolution.

The transformation from the time-domain to the frequency domain is a resultant of multiplying a time signal's response by Euler's complex exponential, denoted $e^{-j\omega t}$ in Euler's Identity below [18]:

$$e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t) \quad (4)$$

where the angular frequency is equivalent to $2\pi f_k$ in the complex exponential in (1), and the corresponding time of interest t is equivalent to $r\delta t$ in the argument of the temporal signal in (1).

As observed in Euler's Identity, multiplying a function by the exponential $e^{-j\omega t}$ amounts to converting a real time signal to complex frequency-dependent function, in which an amplitude and a phase can be extracted, both as functions of frequency. In addition, the reason the frequency index k is limited to one less than half of the number of temporal samples is due to presence of a complex conjugate of the signal. Meaning that, the frequency-dependent absolute value amplitude for k ranging from 0 to $\frac{N}{2} - 1$ is identical, yet mirrored, to that of k ranging from $\frac{N}{2}$ to N, as the former k range represents the absolute value of the positive complex conjugate of the signal, while the latter represents the absolute value of the negative complex conjugate. In this case, only the amplitudes correlating to the positive conjugate are considered while the counter is neglected to avoid redundancy in data.

2.3 Short-time Fourier Transform

For analyzing a stationary temporal signal via DTFT, as stated earlier, it is essential to use the entire temporal data signal (from 0 to $N - 1$) for an input. Accordingly, DTFT is well-suited for the analysis of periodic signals that consist of time-invariant frequency components. In contrast, for signals that contain time-varying frequency components, the STFT is employed. To capture the time-varying amplitudes at various frequencies, STFT

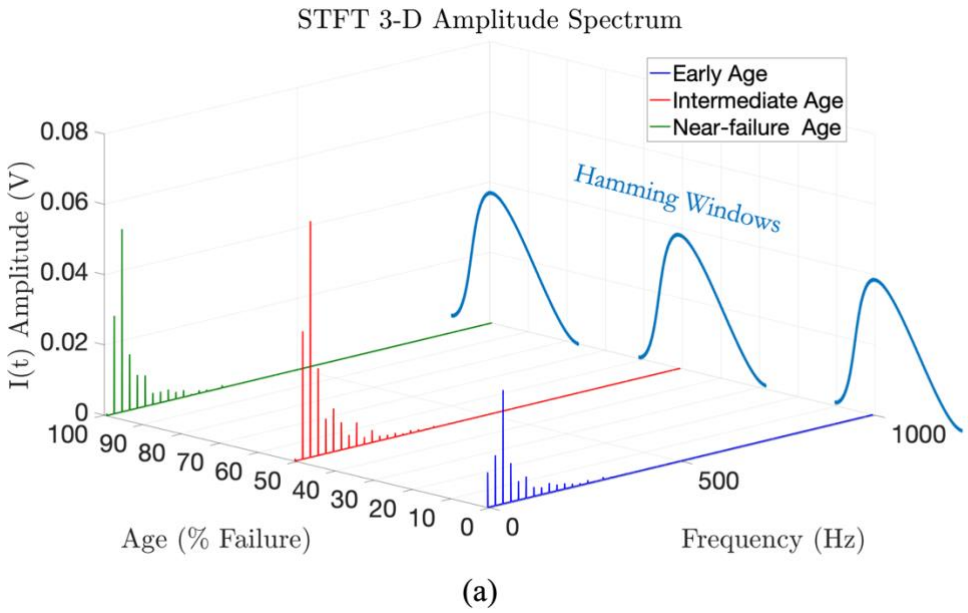
can be defined as the execution of DFT on a signal over multiple segments or windows of the temporal signal rather than implementing it over the whole signal. Implying that, unlike the output of DFT, which is a function of frequency only, the output of an STFT function is a function of both frequency and time. Further, the implementation of STFT obligates the temporal data segment of interest to be multiplied by a window function (i.e. Rectangular, Barlett/Triangular, Hanning, Hamming, Blackman, and Kaiser windows) to alter spectral leakage [19] at the beginning and the end of the data segments. The following two equations preview the underlying mathematics of STFT as well as a hamming window [20]:

$$X[n,k] = \sum_{m=0}^{L-1} x[n+m] * w[m] e^{-\frac{j2\pi km}{N}}, \qquad 0 \le k \le N-1 \tag{5}$$

$$w[m] = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi m}{L}\right), & 0 \le m \le L-1 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

where N is half the of sampling frequency, k is the frequential index, m is the time index relative to the hamming window function, and n is the time index relative to the entire data signal. As noticed in the argument of the input discrete function $x[n+m]$, the temporal signal is slid over a stationary hamming window that always starts with the index m at zero and ends at $L-1$, where L is the number of points forming both the hamming window function and the portion of the data that overlaps with the window function. Moreover, for every two successive data windows, the presence of an overlap region in between is recommended to reduce the negation effect caused by the edges of the window function. Figure 3(a) below demonstrates a sample of the STFT process carried out in this study,

where the amplitude is displayed as a function of age and frequency in a 3-D amplitude spectrum. Besides, Figure 3(b) displays the results of implementing STFT for 100 windows on a 2-D spectrogram, along with a 3-D spectrogram at the right of the figure.



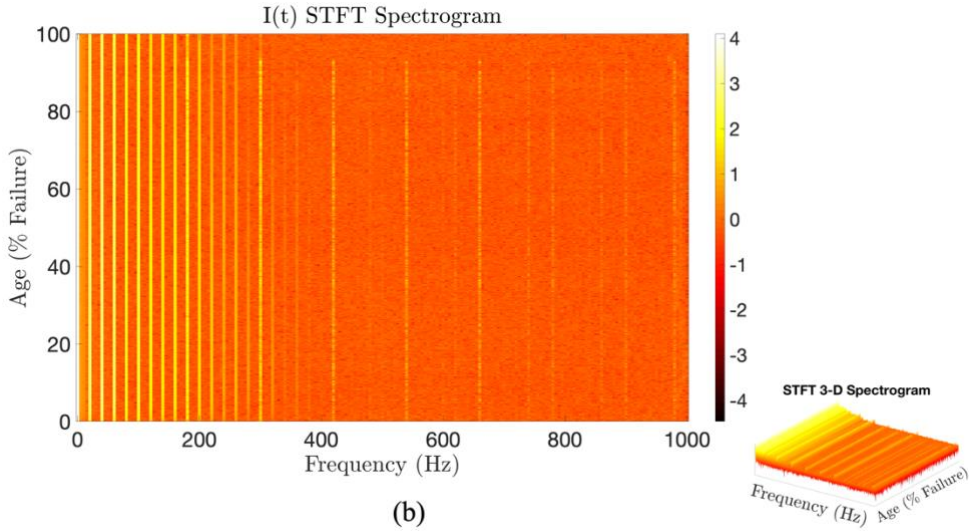


Figure 3(a): STFT amplitude spectrums for the early, intermediate, and near-failure luminescence windows with hamming windows implemented on each of the windows. Figure 3(b): STFT results are plotted on a spectrogram, where the increasing brightness corresponds to increasing amplitude. A 3-D display of the spectrogram is shown on the right.

2.4 Conclusion

For the purposes of this work, the raw luminescence data signal was split into one hundred windows –each window represents luminescence data pertaining to a specific age or health state– with an overlap region comprised of 1% of the total length of a window. Each of these windows were passed through a hamming window prior to the execution of STFT. The reason the Hamming window was chosen over the other window functions is that it never goes to zero, which reduces the negation effect at the edges of the window while guaranteeing negligible spectral leakage. In addition, the choice of a Hamming window, along with the low variability of the luminescence data relative to the number of points forming the signal, allowed for the selection of a minute overlap region, thus reducing computational time. The end resultant of STFT is one hundred light intensity amplitude spectrums that are affiliated with one hundred successive ages for a single composite coupon specimen, integrated into a single spectrogram shown in Figure 3(b). Using the information extracted from this spectrogram, machine learning tools are utilized for discriminating between the spectral peaks at health stages of coupon (distinct ages), therefore enabling estimation of remnant life in EML composite coupon performing SDP. In the coming section, we introduce Support Vector Machines and Logistic Regression, the primary building blocks behind construction of Neural Networks and classification models.

Chapter 3. Logistic Regression

3.1 Introduction: Separating Data of Different Classes

For any classification problem, an algorithm's goal is, first, to learn how to identify, in a given dataset, data points that belong to different predefined categories; hence, when given a new data point, the algorithm is able to assign it to one of these known categories. Logistic Regression (LR) is a supervised statistical learning method that aims to separate the inputted dataset into discrete predefined classes via the construction of an optimized hyperplane –also known as decision boundary or plane– in a manner that is consistent with the provided training examples. To perform LR, three main parameters must be well-defined with the classification problem: I) the number of features, II) the number of possible classes, and III) the number of training examples or number of sample data points per class. The features in a dataset are the independent variables that create the different classes, as data points of features possessing close magnitudes most likely fall under a similar class while those holding afar values are of differing classes. Determining the number of classes or data clusters is uncomplicated when the nature of the problem limits the algorithm to a discrete number of options to choose from, such as these associated with character-recognition and image-recognition algorithms. Contrariwise, a classification problem open to an infinite number of classes is more daunting and requires

a measure of the variability in the dataset to force discretization; such techniques include K-means Clustering [21].

3.2 Implementing Logistic Regression

In this study, an algorithm's ultimate objective, when given input features for different training examples, is to recognize the age or health state of the sample. Hereby, the sample's ages are the preset classes in which an algorithm needs to classify distinct sectors of the dataset to. Likewise, the input features, which are the independent parameters that induce different health states, are the unique peak values that comprise the amplitude spectrums associated with distinctive ages or health states. These peak amplitudes, which are referred to as STFT coefficients, occur at multiples of the actuation frequency, 20 Hz, and discontinues at half of the sampling frequency, 1000 Hz, as observed in any of the amplitude spectrums in Figure 2(c). Hence, for each of the six tested coupons or the training examples affiliated with this work, the algorithm encompasses 49 features, represented by STFT coefficients at frequencies starting from 20 Hz up to 980 Hz in multiples of 20 Hz. In addition, 100 distinct classes, represented by ages between 0% of failure to 100% of failure in increments of 1%, were selected from a nonfinite viable number of classes based on the variance of the temporal data, which dictates how distinguishable a class from an algorithm's perspective.

To expound upon the procedure of developing a linear multi-dimensional decision boundary using multiple training examples, the group of equations below is addressed.

$$G_k(\Theta_k, X) = \begin{pmatrix} g_1 = \theta_0 x_{0,1} + \theta_1 x_{1,1} + \theta_2 x_{2,1} + \cdots + \theta_n x_{n,1} \\ g_2 = \theta_0 x_{0,2} + \theta_1 x_{1,2} + \theta_2 x_{2,2} + \cdots + \theta_n x_{n,2} \\ g_3 = \theta_0 x_{0,3} + \theta_1 x_{1,3} + \theta_2 x_{2,3} + \cdots + \theta_n x_{n,3} \\ \vdots \\ g_m = \theta_0 x_{0,m} + \theta_1 x_{1,m} + \theta_2 x_{2,m} + \cdots + \theta_n x_{n,m} \end{pmatrix} \xrightarrow{\text{yields}} \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_m \end{bmatrix}^T = \begin{bmatrix} \theta_{k,1} \\ \theta_{k,2} \\ \theta_{k,3} \\ \vdots \\ \theta_{k,m} \end{bmatrix}^T * \begin{bmatrix} x_{0,1} & x_{0,2} & x_{0,3} & \cdots & x_{0,m} \\ x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & x_{n,3} & \cdots & x_{n,m} \end{bmatrix} = \Theta_k^{\mathrm{T}} X \qquad (7)$$

$$H_k(G_k) = [h_1 \quad h_2 \quad h_3 \quad \dots \quad h_m] = \frac{1}{1+e^{-G_k}}, \qquad \text{where } \begin{cases} h_j \in (0,1) \\ k = 1, 2, 3, \dots, c \end{cases} \tag{8}$$

$$Y_k = [y_0 \quad y_1 \quad y_2 \quad \cdots \quad y_m] \tag{9}$$

For equation (7), the elements of a column in the matrix X, excluding x_0, represent the n features or coordinates of a training example in an n-dimensional space, where x_0 is a constant that maintains a value of unity as the weight θ_0 is a bias unit that controls the elevation of the hyperplane. Thus, for m-training examples, the matrix X in (7) encompasses all the coordinates that describe the positions of all the training examples present in a space. For any of the equations in the set of linear in (7), it can be observed that by setting g to zero the resultant formula is identical to the standard form equation of a plane of $n - 1$ dimensions in the n-dimensional space.

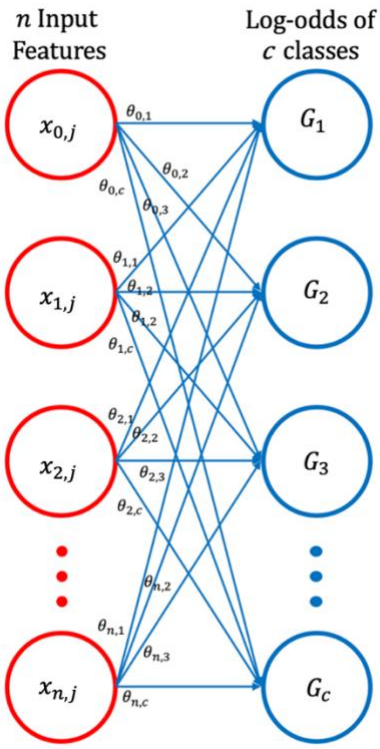


Figure 4: A schematic displaying the weights connecting the features to the log-odds.

The plane described here is the decision hyperplane that isolates the region of interest corresponding to the k^{th} class, where the weights column vector Θ_k, which is initially unknown, contains the characterizing weights that determine the orientation of the hyperplane in space. The index k associated with the weights vector Θ_k determines the index of the class that the hyperplane intends to isolate out of c available classes. That is, to segregate all c classes, c different hyperplanes characterized by c distinct weight vectors are needed to fully apply LR. To generate these c decision boundaries, the algorithm can either utilize equations (7)-(9) and iterate them over c classes or take advantage of the augmented form of the equations proposed below, which is represented schematically in Figure 4.

$$G(\Theta,X)=\begin{bmatrix}G_1(\Theta_1,X)\\G_2(\Theta_2,X)\\G_3(\Theta_3,X)\\\vdots\\G_c(\Theta_c,X)\end{bmatrix}=\begin{bmatrix}g_{1,1}&g_{1,2}&g_{1,3}&\cdots&g_{1,m}\\g_{2,1}&g_{2,2}&g_{2,3}&\cdots&g_{2,m}\\g_{3,1}&g_{3,2}&g_{3,3}&\cdots&g_{3,m}\\\vdots&\vdots&\vdots&\ddots&\vdots\\g_{c,1}&g_{c,2}&g_{c,3}&\cdots&g_{c,m}\end{bmatrix}=\begin{bmatrix}\theta_{0,1}&\theta_{0,2}&\theta_{0,3}&\cdots&\theta_{0,n}\\\theta_{1,1}&\theta_{1,2}&\theta_{1,3}&\cdots&\theta_{1,n}\\\theta_{2,1}&\theta_{2,2}&\theta_{2,3}&\cdots&\theta_{2,n}\\\vdots&\vdots&\vdots&\ddots&\vdots\\\theta_{c,1}&\theta_{c,2}&\theta_{c,3}&\cdots&\theta_{c,n}\end{bmatrix}*\begin{bmatrix}x_{0,1}&x_{0,2}&x_{0,3}&\cdots&x_{0,m}\\x_{1,1}&x_{1,2}&x_{1,3}&\cdots&x_{1,m}\\x_{2,1}&x_{2,2}&x_{2,3}&\cdots&x_{2,m}\\\vdots&\vdots&\vdots&\ddots&\vdots\\x_{n,1}&x_{n,2}&x_{n,3}&\cdots&x_{n,m}\end{bmatrix}=\Theta^{\mathrm{T}}X \quad (10)$$

$$H(G)=\begin{bmatrix}H_1(G_1)\\H_2(G_2)\\H_3(G_3)\\\vdots\\H_c(G_c)\end{bmatrix}=\begin{bmatrix}h_{1,1}&h_{1,2}&h_{1,3}&\cdots&h_{1,m}\\h_{2,1}&h_{2,2}&h_{2,3}&\cdots&h_{2,m}\\h_{3,1}&h_{3,2}&h_{3,3}&\cdots&h_{3,m}\\\vdots&\vdots&\vdots&\ddots&\vdots\\h_{c,1}&h_{c,2}&h_{c,3}&\cdots&h_{c,m}\end{bmatrix}=\frac{1}{1+e^{-G}},\quad \text{where}\begin{cases}h_{k,j}\in(0,1)\\j=1,2,3,\dots,m\\k=1,2,3,\dots,c\end{cases} \quad (11)$$

$$Y_{ref}=\begin{bmatrix}Y_1\\Y_2\\Y_3\\\vdots\\Y_c\end{bmatrix}=\begin{bmatrix}y_{1,1}&y_{1,2}&y_{1,3}&\cdots&y_{1,m}\\y_{2,1}&y_{2,2}&y_{2,3}&\cdots&y_{2,m}\\y_{3,1}&y_{3,2}&y_{3,3}&\cdots&y_{3,m}\\\vdots&\vdots&\vdots&\ddots&\vdots\\y_{c,1}&y_{c,2}&y_{c,3}&\cdots&y_{c,m}\end{bmatrix},\quad \text{where } y_{j,k}\begin{cases}1 & \text{for a training example in the class of interest}\\0 & \text{otherwise}\end{cases} \quad (12)$$

Comparing (7) and (10), it is perceived that the vector G_k is now one row vector in the matric G in (10). Likely, in (8) and (9), the vectors $H_k(G_k)$ and Y_k are now single vectors in the matrices $H(G)$ and Y_{ref}, respectively. To attain the weights matrix Θ, the algorithm takes advantage of the matrix $G(\Theta,X)$, namely the matrix logarithm of odds or the log-odds matrix. A Log-odds is a number computed for each j^{th} training point that quantifies the location of a training example relative to the k^{th} decision boundary. To compute the log-odds, the matrix Θ is assigned arbitrary values and then multiplied by the features' matrix X. The log-odds holds a positive value if the inputted training example lies inside the area of interest, a negative value when the point neither resides on the hyperplane nor in the region of interest, or zero if inhabiting the hyperplane. Initially, the computed log-odds, like their control weights, are random and far from optimized. To optimize these weights, the log-odds are compared with binary reference values populating the matrix Y_{ref}. However, as the log-odds can possess any real number magnitude, a comparison process with binary values obligates the normalization of the log-odds to values between 0 and 1. To normalize the log-odds, the logistic function in (11) is utilized to constrain the

output to magnitudes between 0 and 1, where each element in $H(G)$ yields the probability of locating the corresponding j^{th} training example in the k^{th} region of interest. Implying that, a training point with corresponding a large positive log-odds value, indicating its presence inside the bounded zone of interest, yields a probabilistic value $h_{k,j}$ that is close to 1. Opposingly, a training example that is far from the intended bounded zone is very improbable to be found inside the zone of interest, returning a probability of approximately 0. And, as expected, a sample point on the bounds of the region of interest will result in a logistic output of 0.5. After computing the probabilities affiliated with each training example and before proceeding to the comparison process, the declaration of the matrix Y_{ref} is necessary. The reference probabilities matrix Y_{ref} is predefined for a user who is aware of the classes that each training example is affiliated to. In Y_{ref}, to indicate that the j^{th} sample point belongs the k^{th} class, the element $y_{k,j}$ in the k^{th} row and the j^{th} column is set to 1, while the rest are given 0. Hence, by implementing this concept to every training example, Y_{ref} becomes a reference matrix that holds information about the mapping of every data point to its predetermined class. Then, after completely defining Y_{ref} and computing the matrix H, these two matrices are compared vector-by-vector via the succeeding logistic cost function.

$$J(\Theta) = -\frac{1}{m}\left[\sum_{k=1}^{c}\sum_{j=1}^{m} y_{k,j} * \log\left(h_{k,j}\right) + \left(1 - y_{k,j}\right) * \log\left(1 - h_{k,j}\right)\right] \qquad (13)$$

In this cost function, for a reference probability $y_{k,j}$ valued at 1, the term to the right of the addition sign goes to zero and the error is reduced to the natural logarithm of the probability $h_{k,j}$. Thus, for as the probability approaches 1, the error goes to zero. Meaning that, for a

sample point inside the region, which yields a probability value close to 1, the error is reduced; otherwise, the error is maximized for a point outside the area of interest with a probability close to zero. Conversely, for a reference probability $y_{k,j}$ valued at 0, the error is maximized for a data point inside the region of interest and minimized for a point outside. The cost function is iterated and averaged over the entire dataset with the error being accumulated.

3.3 Cost Calculation and Weights Updating Using Gradient Descent

Initially, due the randomness of the weights in Θ, a hyperplane of a random orientation and elevation is defined. As a result, significant discrepancies occur between the reference probabilities matrix Y_{ref} and the probability matrix H, increasing the accumulated cost calculated from (13). To optimize the decision boundary, the error must be minimized, which was done by taking the derivative of the error with respect to the weights and equating it to zero. The optimization operation implemented on a logistic cost function is gradient descent, given by the following iterative process:

1) Define: $\frac{\delta J}{\delta \theta_{:,i}} = \frac{1}{m}\left[\sum_{j=1}^{m}((h_{:,j} - y_{:,j}) * x_i^T)\right]$, where $i = 1, 2, 3, \ldots, n$ (14)

2) Update: $\theta_{:,i} \coloneqq \theta_{:,i} - \alpha \frac{\delta J}{\delta \theta_{:,i}}$ (15)

3) Compute new cost: $J(\Theta) = -\frac{1}{m}\left[\sum_{k=1}^{c}\sum_{j=1}^{m} y_{k,j} * \log(h_{k,j}) + (1 - y_{k,j}) * \log(1 - h_{k,j})\right]$

4) Repeat steps 1 to 3 until convergence occurs.

In gradient descent, a derivative with respect to each of the n-weights is calculated, corresponding to the path of steepest descending slope from the calculated error. The colon subscript in many of the parameters denotes that the operation is done across all rows. The

parameter α, known as the learning rate, is user-defined parameter that governs the step size of error minimization. Manipulation of the learning rate is key because the error may either diverge if the allotted learning rate is oversized. Opposingly, an undersized learning rate may cause a delayed convergence, which amplifies the computational cost.

After executing the above procedure for an adequate number of iterations, usually until the deduction in the error per iteration is insignificant, the weights should be optimized, inducing a best-fit hyperplane that secludes the data pertaining to the class of interest.

3.4 A Summary of the Steps of implementing LR and a Conclusion

In summary, the following course of action lists the steps of applying logistic regression on a dataset:

1) Knowing the features that cause the major variability in the dataset: Define the matrix X of size n-by-m to match the form expressed in (10).
2) Knowing the possible number of classes and total number of training examples: Define Y_{ref} of size c-by-m as shown in (12).
3) Not knowing the optimal orientations of the decision-boundaries: arbitrarily assign the weights matrix as expressed in (10)
4) Multiply the features matrix with the weights' matrix to calculate the log-odds matrix observed in (10)
5) Insert all the log elements as an input to the logistic function defined in (11).
6) Compare the calculated probabilities matrix $H(G)$ with the reference probability matrix Y_{ref} using the cost function defined in (13)

7) Aim to minimize the cost function by first computing the derivative of the cost term with respect to each weight element as seen in (14)
8) Finally, update the weights' matrix based on the calculated derivative as shown in (15)
9) Repeat steps 4 – 8 until the cost function is minimized to an acceptable quantity.

Figure 5 displays logistic regression implemented on the dataset attained from STFT, considering only two out of the 49 available features: Amplitudes at 20 Hz and Amplitudes at 40 Hz. In this plot, a decision boundary is generated to isolate training examples that correspond to the class of 90-100% failure. In Figure 5, a 2-D domain is manifested along with the equation governing the 1-D decision boundary orientation.

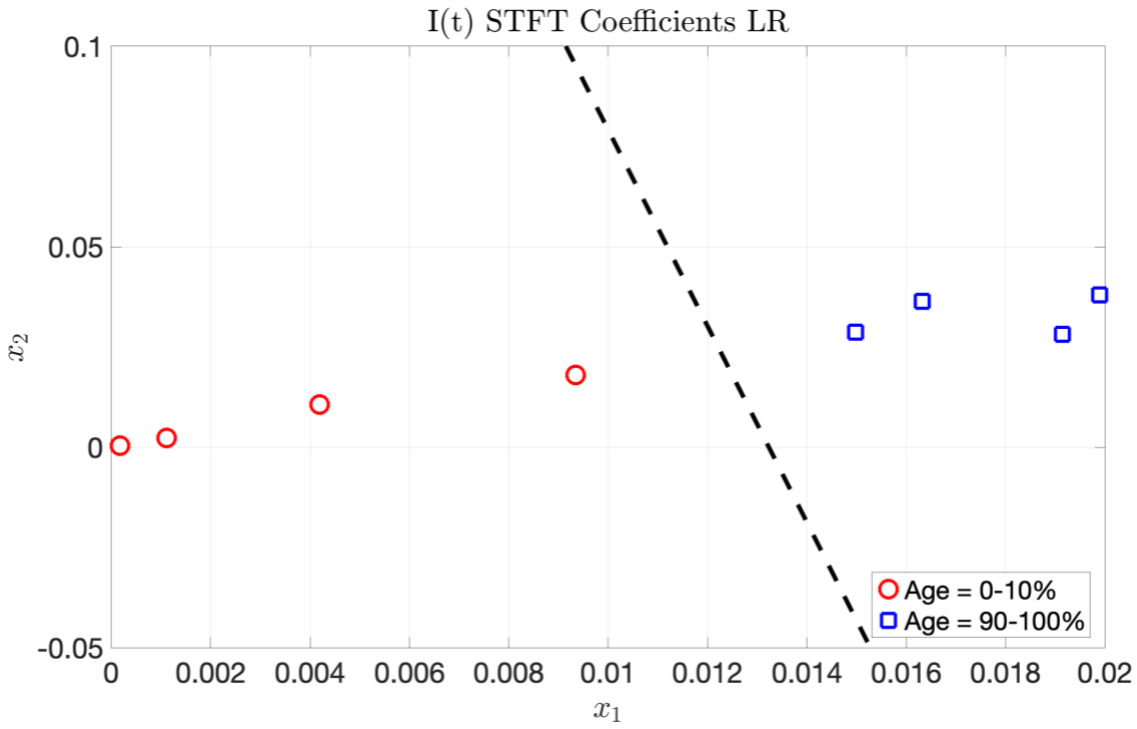


Figure 5: A 2-D hyperplane separating data of early age and near-failure age

Additionally, when the equations in (10) are rearranged in terms of x_2, it is observed that the two computed parameters in Figure 5, 0.324 and -24.48, are both replacing $\frac{\theta_0}{\theta_2}$ and $\frac{\theta_1}{\theta_2}$, respectively. The optimization process can be visually interpreted by observing Figure 6(a) and 6(b), which display the cost decaying with increasing iterations as well as the cost hitting a minimum value when plotted against the two weight parameters' ratios on a contour, respectively.

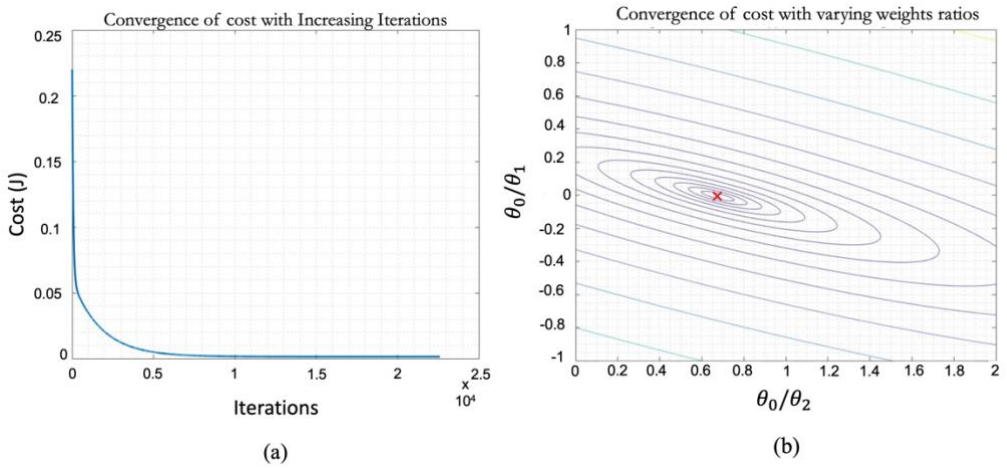


Figure 6(a): The cost associated with the weights matrix drops as iterations increase if a global minimum is found.

Figure 6(b): Hitting an absolute minimum on a surface constructed from parameters $\frac{\theta_0}{\theta_2}$ and $\frac{\theta_1}{\theta_2}$, shown on a contour.

Chapter 4. Artificial Neural Networks

4.1 Introduction: An Overview of Artificial Neural Networks

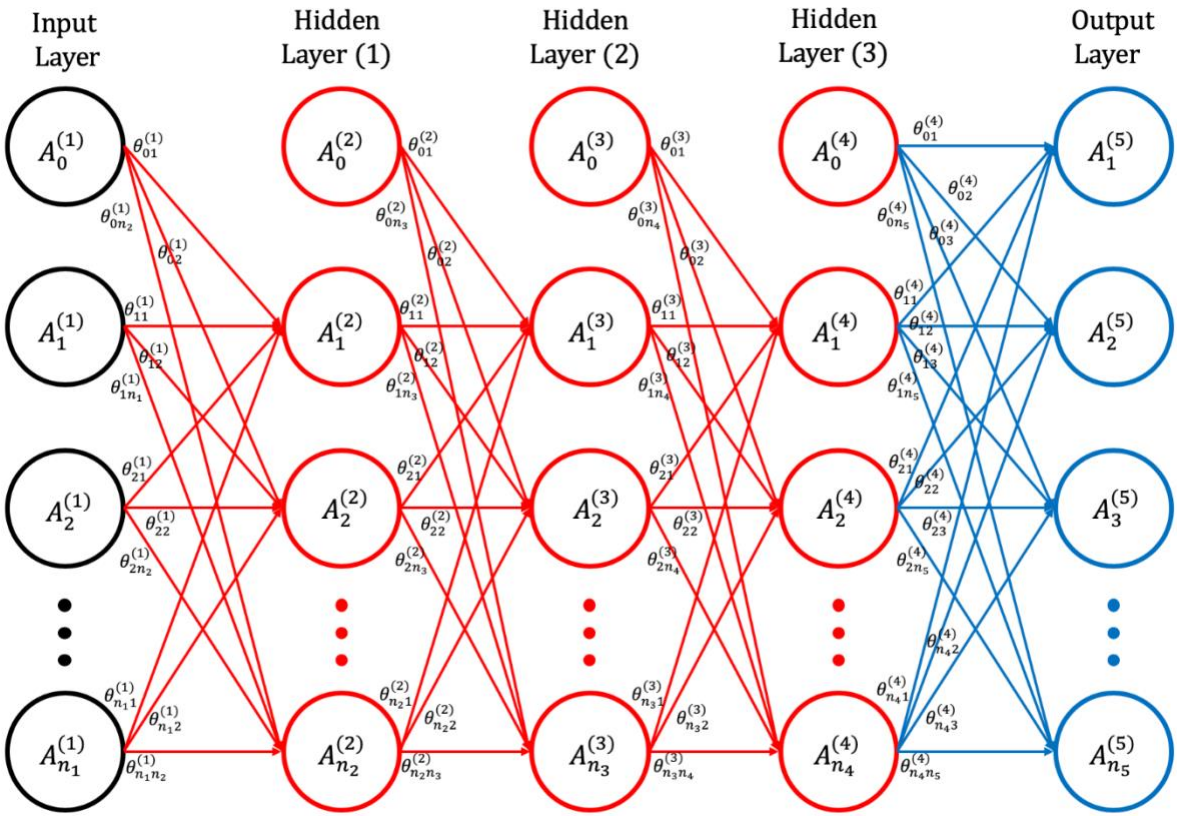


Figure 7: A schematic of a generic ANN architecture

It is perceived from Figure 4 that constructing a single 2-D decision boundary was adequate for segregating one class of data from the other, concluding that the shown dataset as *linearly separable* [22]. The term *linearly separable* is applicable to datasets that can be

separated by a linear hyperplane. In reality, many problems, including the reported SDP problem, involve more complex datasets that are not *linearly separable*, consequently, demanding a more complex form of a hyperplane. Two ways to achieve intricate hyperplanes are nonlinear Logistic Regression and Artificial Neural Networks, where the former relies on directly constructing a nonlinear hyperplane or surface [23][24], while the latter superimposes multiple linear hyperplanes to achieve complexity. In the previous section, in Figure 4, a diagram was introduced as a visual apprehension for the equations in (10). This diagram can be further augmented to the generic schema displayed in Figure 7 above, which is the universal structure of an Artificial Neural Network. In this structure, the insertion of each hidden layer is paralleled by the superposition of an additional hyperplane to the already present hyperplane between the input and the output layers, depicted in Figure 7. Accordingly, LR can be deemed as an ANN that lacks any hidden layers.

An Artificial Neural Networks (ANNs) is a statistical learning tool comprised of interconnected computational units operating in parallel called nodes or neurons. ANNs have been prominently implemented in problems entailing prediction, pattern classification, function approximation, control systems and more [24]-[26]. According to Gardner *et al*. [25], the Neural Networks' taxonomy can be grouped into two chief architectures: I) Feed-forward Networks, which encompass single-layer perceptron, multilayer perceptron, and radial basis function networks, and II) Feedback Networks, which include Competitive networks, Hopfield networks, and ART models. For this work, the focus is on a multilayer perceptron, which is another commonly used term to refer to

an ANN. As portrayed in Figure 7, An ANN is comprised of three major components, an Input Layer, one or more Hidden Layers, and an Output Layer. The Input Layer encompasses n_1 neurons with each neuron representing an independent variable or feature of the input dataset, except the bias neurons $A_0^{(l)}$, which always hold a value of 1. In Figure 7, a neuron is represented as an encircled symbol $A_i^{(l)}$, where the subscript i denotes the unit number while the superscript (l) identifies the number of the layer hosting the neuron. The output layer, on the other hand, is the layer that holds the ANNs' computed decision for a given training example in regards of its class affiliation. Resembling the probabilities matrix $H(G)$, the output layer is an c-by-m matrix. And, finally, the hidden layers, as mentioned earlier, are the additional decision boundaries that make an ANN more suited for *linearly inseparable* datasets.

Prior to the ANN algorithm development, the architecture of the ANN, which outlines the number of nodes present in each layer as well as the number of hidden layers, was first considered. For the number of neurons forming the input layer, it is usually determined based on the number of independent variables constituting a dataset. However, as it can be deduced from Figure 2(c), most of the STFT coefficients, particularly those beyond 80 Hz, do not contribute significantly to the variability of the dataset, therefore, proposing the use of a dimensions' reduction technique. In this study, Principal Component Analysis (PCA) was practiced, reducing the features of the input layer from 49 to 4 STFT coefficients, therefore, shrinking computational cost. PCA is a method that orients the coordinate axes of the dataset in a way that maximizes variability in data, thus identifying features of minimal variability. And since coordinate axes can be configured in infinitely

many ways, the principal axes are found using eigenvalues and eigenvector–more details can be found in Wold [27]. For the hidden layers number, no analytical model exists in determining the ideal number of hidden layers. However, many problems can be resolved using a maximum of one or two hidden layers [28], which is the reason behind using only one hidden layer for SDP implementation. Moreover, the number of hidden layer neurons can be estimated using methods as those highlighted by Cai *et al.* [29] and Rivals [30] and. However, given that optimizing the solution is not a primary focus in this work, only the "rule-of-thumb" guidelines proposed by Karsoliya [28] were followed along with the execution of trial-and-error approach over different number of hidden neurons, bringing about 6 neurons. Finally, for the output layer neurons, which should be congruent to the number of discrete predefined classes, the output layer was selected to encompass only 100 neurons corresponding to 100 distinguishable structural ages. Since SDP is naturally associated with a non-discrete class quantity, the choice of 100 output neurons was purely built upon trial and error. The schematic of the final ANN architecture used for SDP is manifested in Figure 8 below.

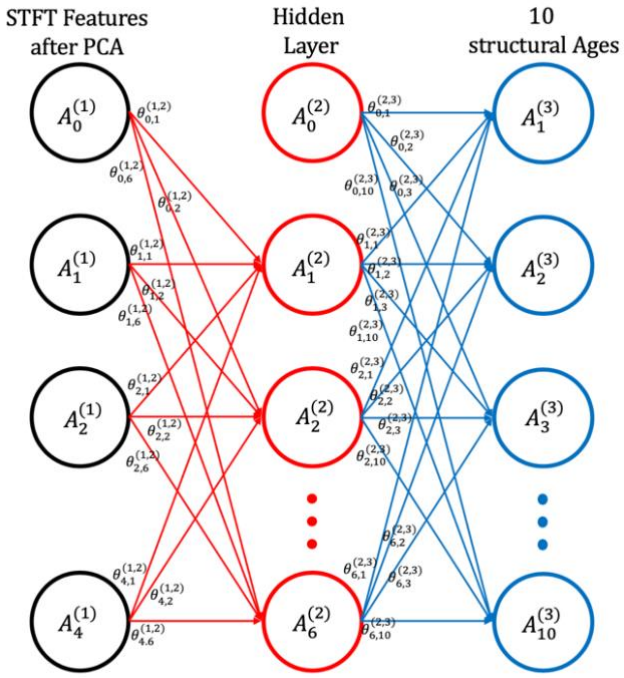


Figure 7: Final SDP ANN architecture, with layers of 4 neurons, 6 neurons, and 100 neurons, respectively.

The way the ANN executes the classification process is broken down in to two main phases: a training phase and a testing phase. The training phase is the stage when the ANN learns data classification based on a given array of training examples, where the class that each training example belongs to is also given. From another perspective, the training phase is the period when the weights matrices are being optimized to configure the hyperplanes on to the desired positions and orientations. In contrast, during the testing phase, which may not precede training, the ANN tests its trained weights by examining its accuracy in correctly classifying a newly fed sample point to one of the predefined classes. In this study, data from five out of the six mechanically tested coupons where categorized as

training data, while data from the remnant coupon was left for testing, inducing a 5-to-1 Train/Test data split as commonly referred to in literature.

4.2 Forward Propagation

For the training phase, the two headlining processes are: forward propagation and back propagation. In forward propagation, the algorithm aims to populate every node in the hidden and output layers with probabilistic values. This can be accomplished using the following steps:

1) Design the ANN by choosing the number hidden layers and number of nodes per hidden layer–consider incorporating a bias unit in all layers, excluding the output layer. Unlike that of the hidden layers, the input and the output layers' dimensions are not open to freedom of design.
2) Define the reference probability matrix Y_{ref} of size c-by-m, with each training example column valued at zero, except for the element at the k^{th} row, referring to its presence in the k^{th} class.
3) Define the input layer matrix $A^{(1)}$ by feeding all training data in the form of m-by-n_1 matrix, each of the m rows containing $n_1 - 1$ features and a bias unit.
4) Define arbitrary weights matrices between each two consecutive layers, where the dimensions of the weights' matrix can be generalized to be $(n_l + 1)$ rows and (n_{l+1}) columns, where l is the index of the source layer and n is the number of non-bias neurons.
5) Populate each neuron in the hidden and output layers by propagating the values held by the input layer neurons. This is applied by first multiplying the input

layer matrix $A^{(1)}$ by the weights' matrix $\Theta^{(1,2)}$, which results in matrix $A^{(2)}$. Matrix $A^{(2)}$ is then normalized using the logistic function in (11). Then, the bias raw vector $A_0^{(2)}$ is retained populated with 1.

6) The resulting layer $A_0^{(2)}$ is then treated as a new input layer and the same procedure in step 4 is repeated until the output layer is computed.

After implementing forward propagation, every element in the output layer matrix will hold the probability of locating a training example in a specific class. Similar to observations from LR, these starting probabilities are anticipated to highly mismatch the reference probabilities, requiring a comparison process that quantifies the error between the output layer and the reference matrix.

4.3 Back Propagation

For an ANN, the counterpart of the comparison process introduced in LR is called back propagation. To execute back propagation, the procedures and formulae below were employed:

1) The error between the output and the reference matrices was computed using the cost function below:

$$J(\Theta^{(L-1,L)}) = -\frac{1}{m}\left[\sum_{k=1}^{c}\sum_{j=1}^{m} y_{k,j} * \log(h_{k,j}) + (1-y_{k,j}) * \log(1-h_{k,j})\right] + \left[\frac{\lambda}{2m}\sum_{l}^{L-1}\Theta^{(l,l+1)}\right] \quad (16)$$

where the subscript L prescribes the total number of present layers. The additional term in this equation, compared to (13), is a regularization term used to reduce overfitting. For the regularization term, all the weights between layers

l and $l+1$ are summed together and regularized using the user-defined regularization constant λ.

2) The gradient associated with each layer was calculated based on the cost between the output and the reference matrices using the following equations:

$$\delta^{(L)} = A^{(L)} - Y_{ref} \quad (17)$$

$$\delta^{(L-1)} = (\Theta^{(L-1)})^T * \left(\delta^{(L)} \cdot A'^{(L-1)}\right), \text{ where } A'^{(L-1)} = A^{(L-1)} \cdot (1 - A^{(L-1)}) \quad (18)$$

where each of these gradients is a matrix of c rows and n columns.

Equation (18) is implemented sequentially until the input layer is reached.

The regularization term in (16) was utilized as, in many cases, after the discriminating hyperplanes are well-fitted to their intended classes, testing the ANN for newly added samples yields low accuracies. This concludes that although the algorithm is capable of distinguishing sample points that have been used for its training, it is unable to distinguish a point that it has never crossed before. This statistical phenomenon is referred to as overfitting; overfitting arises when the constructed hyperplanes enclose their corresponding data cluster in a very tight and inflexible manner, so that any freshly introduced point is less likely to fall inside its intended class. Regularization aids in reducing overfitting by controlling the magnitudes of the weights of a hyperplane. As perceived in (16), the regularization constant λ , which was chosen to be 0.01, is picked by the user to manipulate the cost in a manner that reduces the magnitudes of these weights, thus leading to less convoluted hyperplanes.

For the calculated gradients in (17) and (18), the cost is based only on the output layer's results as the only provided reference is linked to the output layer. For the rest of the layers,

the derivative of each antecedent layer $A'^{(L-1)}$ is used to propagate the cost backwards until the input layer is reached, hence the name back propagation. As mentioned earlier, back propagation only propagates the cost, yet it does not update the weights value based on the propagated cost. To minimize the cost and update the weights, a MATLAB outsourced multi-variate regression function, named `fmincg()` [31]. The function accepts the to-be-minimized function code, initial guesses for the function's inputs, as well as the function termination condition (refer to Rasmussen (2002) for the available conditions) as an input.

4.4 A Summary of ANN implementation

To visualize the logical flow of a neural network, a flow chart of the ANN algorithm used is provided in Figure 9. The logic of the flow in this flow chart follows the order of ANN implementation steps discussed in the previous sections, where forward is first initiated and back propagation is applied until the cost is minimized.

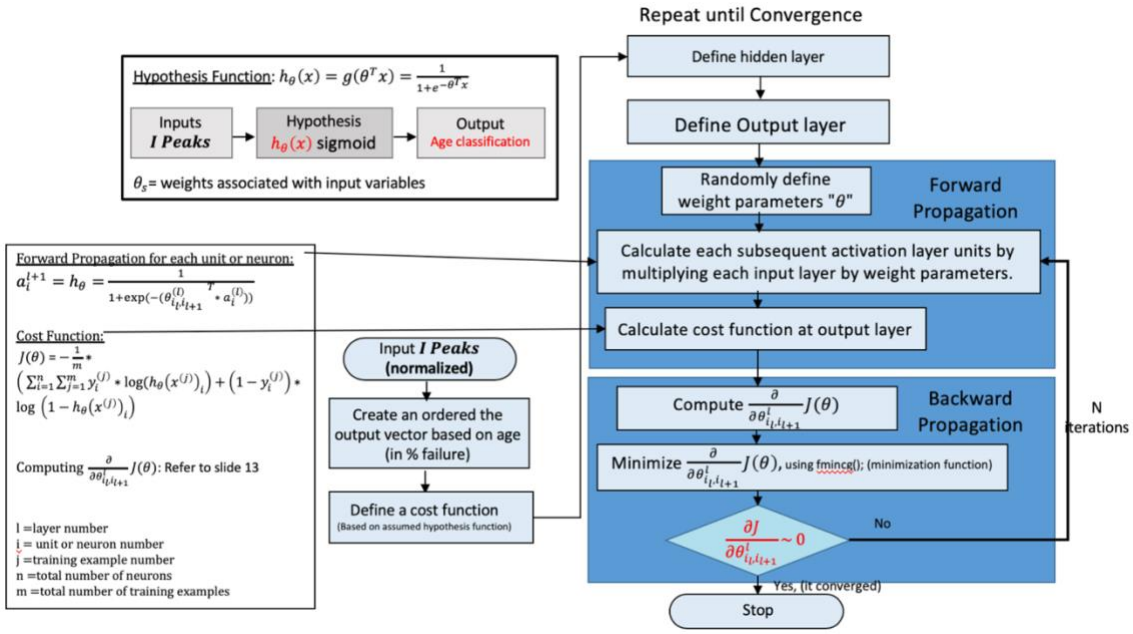


Figure 8: A flow chart displaying the flow in information in forward and back propagation

For the purposes of this project, the inputted arguments were the cost function, randomly initiated weights and 10,000 iterations before termination, respectively. After executing 10,000 iterations, the output was the optimized weights for all hyperplanes pertaining to the 100 structural ages' choices. These weights, which were trained using data from 5 samples, were tested on the remaining sample.

Chapter 5. Results and Conclusion

5.1 Testing Phase Results Using Training and Testing Data

The parameter that measures the success of this algorithm, as mentioned earlier, is its accuracy in detecting the class associated with an added point after tuning the ANN's weights. To test an ANN's performance, firstly, the features of the new sample point are forward propagated through multiplying them by the weights matrices between each two consecutive layers, producing an output layer column vector loaded with probabilities. The index of the maximum probability is the same as the index of the class that the algorithm predicts the point belongs to. Meaning that, if the highest probability in the output vector is that of the k^{th} element, then the algorithm predicts that this sample point belongs to the k^{th} class out of c classes. Repeating this procedure for all testing samples generates the probabilities of finding each point in a class, thus testing the algorithms predictive quality. Accordingly, to compare the predicted ages with the actual coupon ages of the given data: we define accuracy as follows:

$$Q = \left(1 - \frac{Age_{actual} - Age_{predicted}}{Age_{actual}}\right) * 100\% \qquad (19)$$

Accuracy was computed using the training data; that is, the algorithm used the same data for training as for testing. The obtained results are summarized in the histogram in Figure 10.

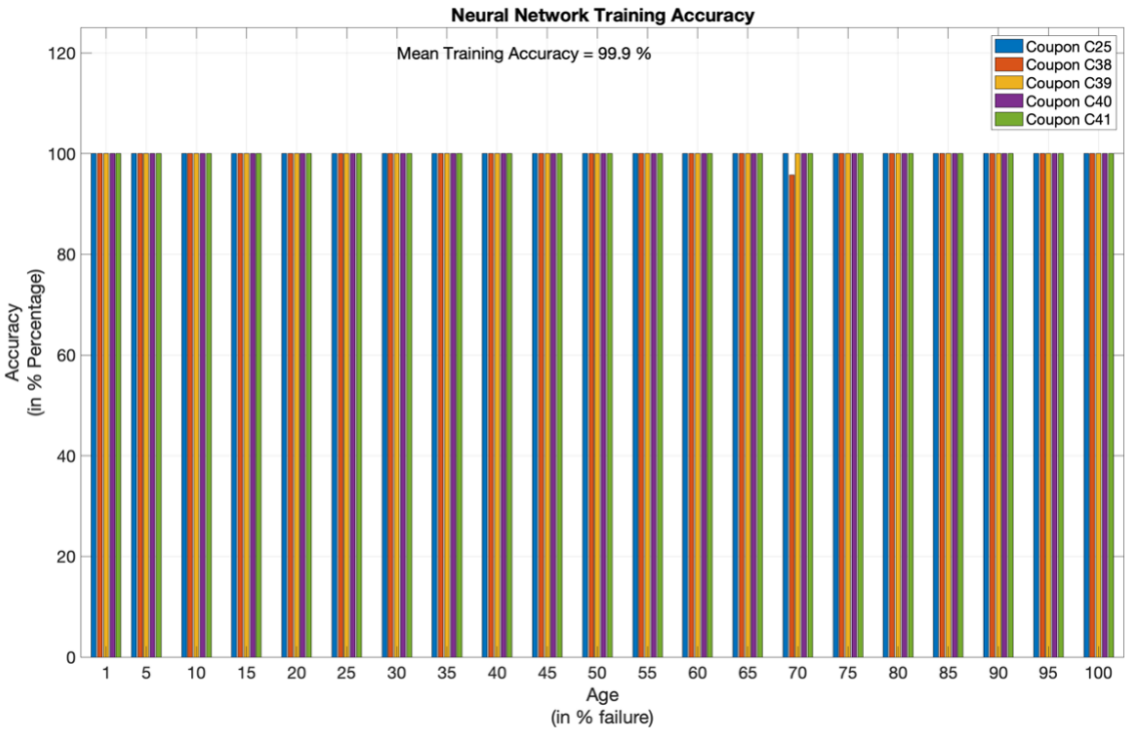


Figure 9: Testing Accuracy using the training data as an input

It is observed that the ANN was very precise in identifying points that were familiarized to the ANN during the training phase. However, as concluded from Figure 11, which displays the accuracy histogram obtained from inputting unfamiliar testing data, the algorithm achieves poor accuracies when assessed against data points that it never encountered before. The reason the mean of the accuracies, labeled in Figure 11, is below zero is that the definition of accuracy does not include an absolute value, allowing for the possibility of negative values.

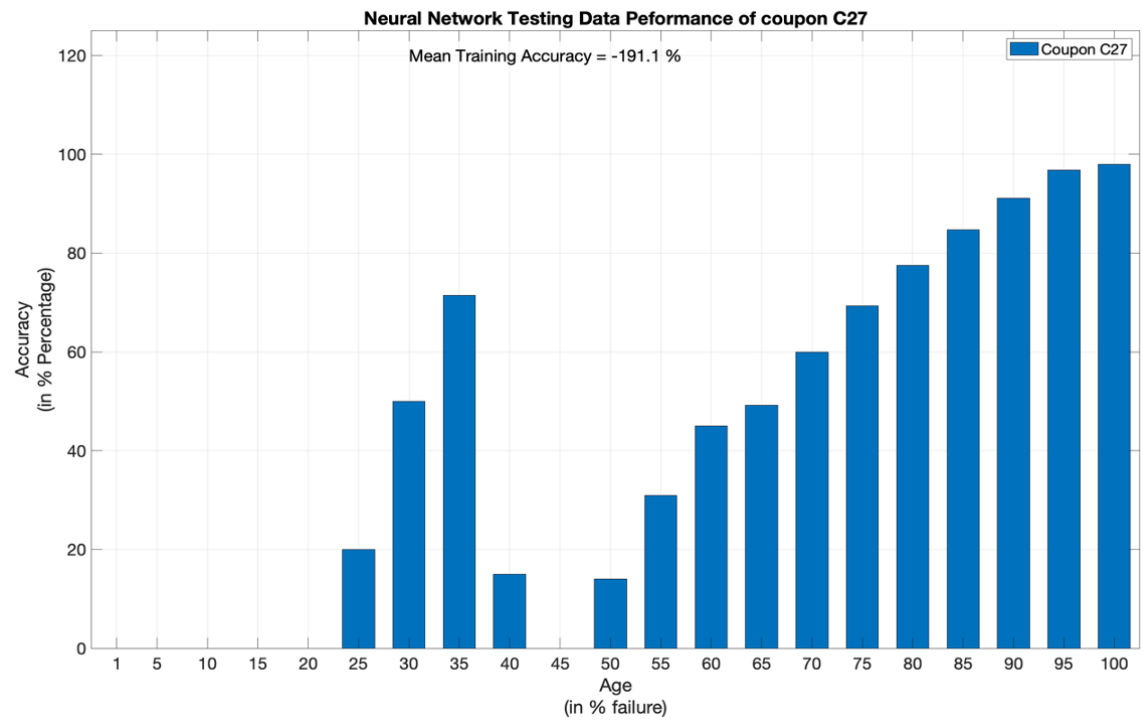


Figure 10:Testing Accuracy using the testing data as an input

5.2 Conclusions and Limitations

This concludes, as stated earlier, the occurrence of overfitting. Albeit using regularization as well as PCA to lessen the likelihood of confronting an overly fit set of hyperplanes, the reason behind overfitting is likely to be intrinsically intertwined with the training data. That is, firstly, the training data is quite scarce, including only data from five samples, which underprepares the algorithm for generalizing its learned prediction to novel data. For most machine learning and data analytical approaches, the performance of the algorithm is strongly proportionate to the number of samples used; nonetheless, for SDP, acquisitioning more data requires the testing of more coupons, which is extremely impractical when each coupon takes tens of hours to accomplish failure. Furthermore, the

five training coupons, in spite of sharing the homogenous fabrication materials, methods, and loading parameters, were of different geometries as well as were subject to different storage conditions. This inhomogeneity in the sample pool altered the captured light response for each training example, which made it troublesome for the algorithm to comprehend a generic trend or pattern in the input data. And in the lack of generic patterns, the algorithm minimizes the cost by overfitting the decision boundaries rather than actually learning how to discriminate between different patterns. Additionally, as claimed earlier, ANNs are generally well-suited for a quantized set of classes, which is not the case here; however, the fact that the algorithm was able to distinguish between classes that were not originally discrete with high accuracies, when the training data was fed in, proves that the algorithm is very likely to distinguish between classes of slight variations of the cost was adequately minimized. In addition to accuracy-related caveats, one limitation to note is the algorithm restriction to cyclic loading data; nevertheless, the logic of ANN and transforming SDP to a classification problem is still applicable to broader forms of loading. Another constraint, which is associated with EML composites, broadly speaking, is output light intensity's dependence on high strain and strain rate, implying that material with relatively low strains and strain rate, such as metal-matrix and ceramic-matrix composites, do not exhibit intense luminescence when fused with ML particles.

Looking at the bright side of the spectrum, it is necessary to mention that this is a pioneer attempt to integrate ANNs with smart material composites in efforts of implementing SDP using a purely data-based approach. Through this work, we lay the foundation of such novel technology that relies on utterly no knowledge associated with

the root physics of the system. Moreover, this attempt has proven to be very promising if given a nonfaulty dataset of identical samples. In addition, the logical and mathematical layout of the algorithm is very flexible regardless of the input samples' geometries or material. Meaning that, the currently reported foundation of teaching an ANN how to classify data is applicable to any geometry or material, despite the production of weights matrices that are unique for every geometry and material combination.

Bibliography

[1] Doebling, Scott W., et al. “A Summary Review of Vibration-Based Damage Identification Methods.” *The Shock and Vibration Digest*, vol. 30, no. 2, Feb. 1998, pp. 91–105., doi:10.1177/058310249803000201.

[2] Farrar, Charles R., and Keith Worden. “Introduction.” *Structural Health Monitoring: a Machine Learning Perspective*, Wiley, 2013.

[3] Lynch, Jerome P., and Kenneth J. Loh. “A Summary Review of Wireless Sensors and Sensor Networks for Structural Health Monitoring.” *The Shock and Vibration Digest*, vol. 38, no. 2, 2006, pp. 91–128., doi:10.1177/0583102406061499.

[4] See, Judi E. “Visual Inspection: A Review of the Literature.” *Sandia National Laboratories Report*, Oct. 2012.

[5] Bond, Leonard J., and Minaz Punjani. "Review of some recent advances in quantitative ultrasonic NDT," in IEE Proceedings A - Physical Science, Measurement and Instrumentation, Management and Education - Reviews, vol. 131, no. 4, pp. 265-274, June 1984.

[6] Sakagami, Takayuki, et al. “Detection of Stress Corrosion Cracking by Sonic-IR Technique.” *Developments in Visual and Other NDE Methods II*, 2007.

[7] Francis, Daniel, et al. “Shearography Technology and Applications: a Review.” *Measurement Science and Technology*, vol. 21, no. 10, 2010, p. 102001., doi:10.1088/0957-0233/21/10/102001.

[8] Sophian, Ali, et al. “Electromagnetic and Eddy Current NDT: a Review.” *Insight*, vol. 43, no. 5, 2001, pp. 302–306.

[9] Kriez, H. “Radiographic NDT — a Review.” *NDT International*, vol. 12, no. 6, Dec. 1979, pp. 270–273., doi:10.1016/0308-9126(79)90086-5.

[10] Yang, Ruizhen, and Yunze He. “Optically and Non-Optically Excited Thermography for Composites: A Review.” *Infrared Physics & Technology*, vol. 75, 2016, pp. 26–50., doi:10.1016/j.infrared.2015.12.026.

[11] Chandra, V.K, et al. “Self-Recovery of Mechanoluminescence in ZnS:Cu and ZnS:Mn Phosphors by Trapping of Drifting Charge Carriers.” *Applied Physics Letters*, vol. 103, no. 16, 2013, p. 161113., doi:10.1063/1.4825360.

[12] Chandra, V.k., and B.p. Chandra. “Dynamics of the Mechanoluminescence Induced by Elastic Deformation of Persistent Luminescent Crystals.” *Journal of Luminescence*, vol. 132, no. 3, Nov. 2011, pp. 858–869., doi:10.1016/j.jlumin.2011.09.054.

[13] Gupta, Sujasha, et al. “Structural Health Monitoring of Composite Structures via Machine Learning of Mechanoluminescence.” *ASME 2019 Conference on Smart*

Materials, Adaptive Structures and Intelligent Systems, 2019, doi:10.1115/smasis2019-5697.

[14] Zhong, Rumian, et al. “A Damage Prognosis Method of Girder Structures Based on Wavelet Neural Networks.” *Mathematical Problems in Engineering*, vol. 2014, Apr. 2014, doi:10.1155/2014/130274.

[15] De Lautour, Oliver Richard, and Piotr Omenzetter. “Prediction of Seismic-Induced Structural Damage Using Artificial Neural Networks.” *Engineering Structures*, vol. 31, no. 2, Feb. 2009, pp. 600–606., doi:10.1016/j.engstruct.2008.11.010.

[16] Krishnan, Srivatsava, and Vishnubaba Sundaresan. “Elastomer Health Monitoring Utilizing Elastico-Mechanoluminescence From Impregnated ZnS:Cu.” *Volume 2: Modeling, Simulation and Control of Adaptive Systems; Integrated System Design and Implementation; Structural Health Monitoring*, 2017, doi:10.1115/smasis2017-3965.

[17] Figliola, Richard S., and Donald E. Beasley. *Theory and Design for Mechanical Measurements*. fifth ed., Wiley, 2011.

[18] Jones, Timothy W. “Euler's Identity, Leibniz Tables, and the Irrationality of Pi.” *The College Mathematics Journal*, vol. 43, no. 5, 2012, pp. 361–364., doi:10.4169/college.math.j.43.5.361.

[19] Lyon, Douglas. “The Discrete Fourier Transform, Part 4: Spectral Leakage.” *The Journal of Object Technology*, vol. 8, no. 7, Nov. 2009, pp. 23–34., doi:10.5381/jot.2009.8.7.c2.

[20] “Chapter 10: Fourier Analysis of Signals Using the Discrete Fourier Transform.” *Discrete-Time Signal Processing*, by Alan V. Oppenheim and Ronald W. Schafer, Third ed., Pearson, 2014.

[21] Jordan, M., et al. “Chapter 9: Mixture Models and EM.” *Pattern Recognition and Machine Learning*, by Christopher M. Bishop, Springer New York, 2006, pp. 424–430.

[22] Jordan, M., et al. “Chapter 4: Linear Models for Classification.” *Pattern Recognition and Machine Learning*, by Christopher M. Bishop, Springer New York, 2006, pp. 179–181.

[23] Salter, Mark A., et al. “Modelling the Combined Temperature and Salt (NaCl) Limits for Growth of a Pathogenic Escherichia Coli Strain Using Nonlinear Logistic Regression.” *International Journal of Food Microbiology*, vol. 61, no. 2-3, Nov. 2000, pp. 159–167., doi:10.1016/s0168-1605(00)00352-4.

[24] Cheng, Bing, and D. M. Titterington. “Neural Networks: A Review from a Statistical Perspective.” *Statistical Science*, vol. 9, no. 1, 1994, pp. 2–54., doi:10.1214/ss/1177010638.

[25] Gardner, Matthew W., and Stephen R. Dorling. “Artificial Neural Networks (the Multilayer Perceptron)—a Review of Applications in the Atmospheric Sciences.” *Atmospheric Environment*, vol. 32, no. 14-15, 1998, pp. 2627–2636., doi:10.1016/s1352-2310(97)00447-0.

[26] Awodele, Oludele, and Olawale Jegede. “Neural Networks and Its Application in Engineering.” *Proceedings of the 2009 InSITE Conference*, 2009, doi:10.28945/3317.

[27] Wold, Svante. “Principal component analysis.” *Chemometrics and Intelligent Laboratory Systems.*, vol. 2, 1987, pp. 37-52, 10.1016/0169-7439(87)80084-9

[28] Karsoliya, Saurabh. “Approximating Number of Hidden Layer Neurons in Multiple Hidden Layer BPNN Architecture.” *International Journal of Engineering Trends and Technology*, vol. 3, no. 6, 2012, pp. 714–717.

[29] Cai, Guang-Wei, et al. “Estimating the Number of Hidden Nodes of the Single-Hidden-Layer Feedforward Neural Networks.” *15th International Conference on Computational Intelligence and Security (CIS)*, 2019, doi:10.1109/cis.2019.00044.

[30] Rivals, Isabelle, and Léon Personnaz. “A Statistical Procedure for Determining the Optimal Number of Hidden Neurons of a Neural Model.” *Second International Symposium on Neural Computation (NCí2000)*, 1999.

[31] Carl E. Rasmussen (2002) fmincg() Multivariate Regression (Version 1) [MATLAB Function]. https://www.mathworks.com/matlabcentral/fileexchange/42770

Appendix A: STFT Computation Algorithm

```
%% Information
%{
 Author = Ahmed Mohamed
 Integrated Systems Lab at the Ohio State University
 Date = March 13th, 2020
 File name = MLPaperCode_1
 Premises = Raw data extraction, raw data processing, plotting needed plots,
 and implementing STFT on processed luminescence (or any other data curve of
 interest)
%}

%{
Files and functions needed:

Functions:
STFT.m
FFT.m
CHECKMAKE.m
fmincg_GIF.m

Files:
All data files named as following: cCouponNumber0FileNumber.mat
For instance: for CouponNumber = 25, FileNumber = 01 ---> fileName = c25001.mat

%}

%% Start
clear
clc
close all
tic                                                         % Starting a clock to
calculate time elapsed during algorithm run
currentFolder = cd;                                         % Saving main Folder path
screenInformation = get(0,'screensize');                    % obtaining screen size
screenWidth = screenInformation(end-1);                     % Screen width
screenHeight = screenInformation(end);                      % Screen height
fontSize = 16;                                              % Font size for all plots

% Saving all directory and path related variables and names
currentPath = pwd;                                          % Saves the current
directory path
backSlashes = strfind(currentPath,'/');                     % Contains all back slashes
in the saved path
parentFolderPath = currentPath(1:backSlashes(end)-1);       % Parent folder path
addpath(genpath(parentFolderPath));                         % Add parent  folder and
all subfolders to path

%% Extracting data
% Coupon related constants
couponNumber = [25 27 38 39 40 41];                        % coupon Identification
number
couponLengths = [0.63 0.425 0.500 0.43 0.425 0.44];        % coupon length (in mm)
couponWidths = [10 10 10 10 10 10];                        % coupon width (in mm)
couponThicknesses = [1.7 1.88 1.88 1.88 1.88 1.88];        % coupon thickness (in mm)
```

```
startPoints = [9972 46960 10570 4900 46960 121440];                % Start point at which
recording of data points started
failurePoint = [226858752 332495404 159760688 103882157 173488331 246576825]; % Failure points
indices chosen manually from raw data plots
slippage = [2 3 2 3 3 1.5];                                        % Experimental slippage
while pulling the coupons (in mm)
sensorSensitivity = 1.2;                                           % Sensor sensitivity
dataTypes = {'I','E','stress','strain'};                           % In case another data type
needs a similar analysis (let's say we want STFT for stress or strain vectors)
desiredDataTypes = {'I'};                                          % Desired data types that
the user wishes STFT data for

% Looping across different coupons
for index = 3:size(couponNumber,2)                                 % To execute on all six
coupons

    clearvars -except tic currentFolder screenInformation screenWidth ...
              screenHeight fontSize currentPath parentFolderPath couponNumber ...
              couponLengths couponWidths couponThicknesses startPoints ...
              failurePoint slippage sensorSensitivity dataTypes ...
              desiredDataTypes index

    cd(parentFolderPath);                                                % set current directory
to parent Folder, which contains the folder that contains all the data files
    MLDataFiles = cd ('Data Files');                                   % Set current directory to
ML Data Files Folder
    file = dir (sprintf('c%d*.mat',couponNumber(index)));              % Extracting all information
about files in directory starting with "c." and of ".mat" type
    filenames = {file.name};                                           % Saving file names in one
cell
    filesNumber = size(filenames,2);                                   % Number of files to extract
data from (subtracting the C%d_10k and C%d_ALL_PEAKS files)

    % Defining experiment related constants
    couponLength = couponLengths(index)*25.4 + slippage(index);     % length of coupon under
actuation (in mm)

    % Data Extraction into files
    for idx = 1:filesNumber

        data = load(sprintf('c%d.mat', couponNumber(index) * 10^3 + idx));             %
loading struct holding all data
        fname(idx) = fieldnames(data);                                                 % Saving
data files' names to fname

        % Loading all data (data file 1 has a special condition compared to
        ...other data file)
        if idx == 1 % Extracting data from file 1
            time (1:size(data.(fname{idx}).X.Data,2)) = data.(fname{idx}).X.Data;        % Time
(sec)
            force (1:size(data.(fname{idx}).Y(1).Data,2)) = data.(fname{idx}).Y(1).Data; %
Force (N)
            dis(1:size(data.(fname{idx}).Y(2).Data,2)) = data.(fname{idx}).Y(2).Data;    %
displacement (mm)
            lum (1:size(data.(fname{idx}).Y(3).Data,2)) = data.(fname{idx}).Y(3).Data;   % ML
luminescence (volts)
        else        % Extracting data from file 2 --> files_number
            time ((size(time,2)+1):size(time,2)+size(data.(fname{idx}).X.Data,2)) =
data.(fname{idx}).X.Data;           % Time (sec)
            force ((size(force,2)+1):size(force,2)+size(data.(fname{idx}).Y(1).Data,2)) =
data.(fname{idx}).Y(1).Data;  % Force (N)
            dis ((size(dis,2)+1):size(dis,2)+size(data.(fname{idx}).Y(2).Data,2)) =
data.(fname{idx}).Y(2).Data;        % displacement (mm)
            lum ((size(lum,2)+1):size(lum,2)+size(data.(fname{idx}).Y(3).Data,2)) =
data.(fname{idx}).Y(3).Data;        % ML luminescence (volts)
        end
    end
```

```
    clear data;             % Removing the struct data for memory clearing
    cd (currentFolder)      % Returning to the currentFolder to access other files

    %% Data fixing and computations

    % Defining constants to fix data start point
    preload = mean(force(1:startPoints(index)));            % Force sensor initial force vector
mean shift
    disZero = mean(dis(1:startPoints(index)));              % Light sensor initial lum vector
mean shift

    % Fixing time vector to start from correct point
    % failure_point = failure_point - time(start_points(index))*20/1e6;      % Shifting failure
point
    time = time(startPoints(index):end);                    % Removing points recorded prior to
experiment start
    time = time - time(1);                                  % Starting from time(1) = zero
    lum = lum(startPoints(index):end);                      % Starting from lum(1) = zero
    force = force(startPoints(index):end);                  % Starting from force(1) = zero
    dis = dis(startPoints(index):end);                      % Starting from dis(1) = zero

    % Defining essential frequencies
    dt = time(2) - time(1);                                 % Change in time or time vector
increments
    sampFreq = round(1/dt);                                 % Sampling frequency 2000 samples
per second
    shakerFreq = 20;                                        % Shaker frequency was set to 20
cycles per second
    luminescenceFreq = 2*shakerFreq;                        % Luminescence frequency is
2*shaker frequency as light is emmited once for a tension and once for compression

    % Computing stress, strain, stress rate, and P
    disp = (disZero - dis)/(sensorSensitivity*tand(5)) - slippage(index);    % Dividing by
sensor sensitivity as well as by tangent of 5 degrees (experiment design)
    strain = (disp/couponLength)*100;                                        % Calculating
normal true strain vector
    stress = force/(couponWidths(index)*couponThicknesses(index));           % Calculating
normal engineering stress vector
    %  stress = stressEng./exp(-0.394*strain);                               % Calculating
normal true stress vector

    % Eliminating post-failure points
    t = time(1:failurePoint(index))';
    I = lum(1:failurePoint(index))';
    stress = stress(1:failurePoint(index))';
    strain = strain(1:failurePoint(index))';
    E = stress./strain;

    clear lum time; % to save memory

    %% Implementing Short-time Fourier Transform on a selected dataVector
    % Saving all desired data in a single cell array
    data{1} = I;
%   data{2} = E; % In case another data vector needs analysis
                 % Note that if another desired data type is needed for
              ...analysis, the variable desiredDataTypes  must be updated

    for dataTypeSelector = 1:max(size(desiredDataTypes)) % Once for Luminescence and once for
if not commented out Modulus

        % Defining inputs for the user-defined STFT functions
        dataTypeSelected = dataTypes{dataTypeSelector}; % This indicates which data vector is
being chosen of the options presented in the previous line
        dataVector =  data{dataTypeSelector};   % Selecting Data vector to be transformed to
the frequency domain using STFT
        dataLength = size(dataVector,1);        % Length of the data vector being inputted to
STFT
```

```
        windowsNumber = 100;                    % Number of windows STFT will divide the data
vector into
        overlapPercentage = 10;                 % Percentage of the window length that overlaps
with the subscequent window

        % Calculating window length given the desired number of windows and the
        ...desired overlap length
        windowLength = floor(max(size(dataVector)/(ceil(windowsNumber)*(1-
(overlapPercentage/100)+(overlapPercentage/100)/ceil(windowsNumber)))));

        % Implementing STFT on the dataVector
        [Amps,Freqs,Phase] =
STFT(dataVector,sampFreq,'window','HammingS',windowLength,'overlapPercent',overlapPercentage);

        %% Extracting Amplitude Spectrum Peaks (FFT Coefficients) and corresponding frequencies
for all windows
        % Creating an array cell of ages, where each array holds FFT at a
        ... specific age

        % Specifying constants needed to find peaks using a for-loop
        firstFFTPeak = shakerFreq/2;
        frequencyIncrement = shakerFreq;
        LastFFTPeak = (sampFreq - shakerFreq)/2;
        harmonicFreqs = [firstFFTPeak:frequencyIncrement:LastFFTPeak]; % Frequencies to plot

        % Finding frequency indices that have the FFT Peaks as their midpoints.
        ... In other words, we are trying to find 10Hz, 30Hz, 50Hz, ..., 990Hz
        ... indices so the peaks at 20Hz, 40Hz, 60Hz, ..., are centered between
        ... these indices. This is important when trying to find the peaks using
        ... max(); function.
        for idx = 1:size(Freqs,2)
            for id = 1:size(harmonicFreqs,2)
               [~,frequencyIndices(idx,id)] = min(abs(harmonicFreqs(id)-Freqs(:,idx)));
            end
        end

        % Finding peaks in all FFT Data
        for idx = 1:size(Amps,2)
            for id = 1:(size(frequencyIndices,2)-1)
                [STFTCoefficients(idx,id),STFTCoefficientsLocations(idx,id)] =
max(Amps(frequencyIndices(idx,id):frequencyIndices(idx,id+1),idx));
                STFTFrequencies(idx,id) = Freqs(frequencyIndices(idx,id) +
STFTCoefficientsLocations(idx,id) -1,idx);
            end
        end

        % Saving STFT Coefficients and corresponding Frequencies into a separate file

        CHECKMAKE('STFT Coefficients'); % A function that checks for the presence of a folder,
                                        ... it only creates a folder with the inputted name
                                        ... if the folder does not exist, else it does nothing.
        STFTCoeffientsFolder = cd('STFT Coefficients');

save(sprintf('Coup_%d_%s_STFT_Coefficients_&_Freqs',couponNumber(index),dataTypeSelected),'STFT
Coefficients','STFTFrequencies');
        cd(currentFolder);



        % If normalization of data is desired
        normalize = false; % change to true if needed, else leave as false

        % Normalization is done based on the largest peak in the amplitude
        ...spectrum
        if normalize
            for idx = 1:size(STFTCoefficients,1)
                NormSTFTCoefficients(idx,:) =
STFTCoefficients(idx,:)/(max(STFTCoefficients(idx,:)));
```

```
            end

            % Saving Normalized STFT Coefficients and corresponding Frequencies into a separate
file
            CHECKMAKE('Normalized STFT Coefficients');
            NormSTFTCoeffientsFolder = cd('Normalized STFT Coefficients');

save(sprintf('Coup_%d_%s_Norm_STFT_Coefficients_&_Freqs',couponNumber(index),dataTypeSelected),
'NormSTFTCoefficients','STFTFrequencies');
            cd(currentFolder);
        end

    end
end

% Looping across different coupons
for index = 1:size(couponNumber,2)
%% ============================== Plots ============================== %%
    %% Figure 1: Plotting Raw Data I with sections
    RawDataPlot_I_stress_strain = figure('Name','Luminiscence, Stress, and Strain Raw Data');
    leftScreenFit = [0 0 screenWidth screenHeight];
    doubleScreenFit = [0 0 screenWidth*2 screenHeight];
    rightScreenFit = [screenWidth 0 screenWidth screenHeight];
    set(RawDataPlot_I_stress_strain,'Position',leftScreenFit);
    set(groot,'defaultLegendInterpreter','default');
    set(gca,'fontSize',fontSize);

    subplot(3,1,1)
    plot(t,I,'Color','k','LineWidth',1.0)
    set(gca,'fontSize',fontSize);
    % title(sprintf('%s (t)',dataTypeSelected));
    xlabel('Time (s)','interpreter','latex');
    ylabel(sprintf('Luminescence %s(t) (V)', dataTypeSelected),'interpreter','latex');
    if normalize
        ylim([0 1.00]);
    end
    grid on

    subplot(3,1,2)
    plot(t,stress,'Color','k','LineWidth',1.0)
    set(gca,'fontSize',fontSize);
    % title('Stress');
    xlabel('Time (s)','interpreter','latex');
    ylabel('Stress $\sigma$(t) (MPa)','interpreter','latex');
    grid on

    subplot(3,1,3)
    plot(t,strain,'Color','k','LineWidth',1.0)
    set(gca,'fontSize',fontSize);
    % title('Strain');
    xlabel('Time (s)','interpreter','latex');
    ylabel('Strain $\epsilon$(t) (unitless)','interpreter','latex');
    grid on

    %  One title for all subplots
    subplotTitle = sgtitle('Luminiscence, Stress, and Strain Raw Data','interpreter','latex');
    subplotTitle.FontSize = 20;

    CHECKMAKE('Figures');
    Figures = cd('Figures');
    %
saveas(RawDataPlot_I_stress_strain,sprintf('Coup_%d_%s_I_stress_strain_Raw_Data.fig',couponNumb
er(index),dataTypeSelected));

saveas(RawDataPlot_I_stress_strain,sprintf('Coup_%d_%s_I_stress_strain_Raw_Data.png',couponNumb
er(index),dataTypeSelected));
    cd(currentFolder);
```

```
    %% Figure 2a: Plotting Raw Data I with sections highlighted in colors
    % Data Extraction for the plots
    t_Windows = [t(1:ceil(windowLength))
t(ceil(size(I,1)/2)+1:ceil(size(I,1)/2)+ceil(windowLength)) t(end-(ceil(windowLength)-1):end)];
    I_Windows = [I(1:ceil(windowLength))
I(ceil(size(I,1)/2)+1:ceil(size(I,1)/2)+ceil(windowLength)) I(end-(ceil(windowLength)-1):end)];

    % Plotting
    RawDataPlot_I_windows = figure('Name',sprintf('Coupon %d %s(t) Raw Data with Specified
Sections',couponNumber(index),dataTypeSelected));
    set(RawDataPlot_I_windows,'Position',leftScreenFit);
    set(groot,'defaultLegendInterpreter','default');

    plot(t,I,'Color','k','LineWidth',1.0,'HandleVisibility','off')
    hold on;
    plot(t_Windows(:,1),I_Windows(:,1),'Color','b','LineWidth',2.0)
    plot(t_Windows(:,2),I_Windows(:,2),'Color','r','LineWidth',2.0)
    plot(t_Windows(:,3),I_Windows(:,3),'Color',[0 0.5 0],'LineWidth',2.0)
    set(gca,'fontSize',fontSize);
    title('Luminesecence Windows at Different Health States','interpreter','latex');
    xlabel('Time (s)','interpreter','latex');
    ylabel(sprintf('Luminescence %s(t) (V)', dataTypeSelected),'interpreter','latex');
    if normalize
        ylim([0 1.00]);
    end
    grid on
    legend('Early Age','Intermediate Age','Near-failure  Age','location','best')

    CHECKMAKE('Figures');
    Figures = cd('Figures');
    %
saveas(RawDataPlot_I_windows,sprintf('Coup_%d_%s_Raw_Data_windows.fig',couponNumber(index),data
TypeSelected));

saveas(RawDataPlot_I_windows,sprintf('Coup_%d_%s_Raw_Data_windows.png',couponNumber(index),data
TypeSelected));
    cd(currentFolder);

    %% Figure 2b: Plotting 5 cycles of I within the sections highlighted in colors
    % Data Extraction for the plots
    CyclesNumber = 5;
    PointsPerCycle = sampFreq/shakerFreq;
    CyclesPoints = CyclesNumber*PointsPerCycle;
    skippedSeconds = 200;
    startPoint = skippedSeconds*sampFreq;

    t_WindowsCycles = [t_Windows(startPoint+1:startPoint+CyclesPoints,1)
t_Windows(1:CyclesPoints,2) t_Windows(1:CyclesPoints,3)];
    I_WindowsCycles = [I_Windows(startPoint+1:startPoint+CyclesPoints,1)
I_Windows(1:CyclesPoints,2) I_Windows(1:CyclesPoints,3)];

    % Plotting
    RawDataPlot_I_windows_cycles = figure('Name',sprintf('Coupon %d Five Cycles of %s(t) Raw
Data of Specified Sections',couponNumber(index),dataTypeSelected));
    set(RawDataPlot_I_windows_cycles,'Position',leftScreenFit);
    set(groot,'defaultLegendInterpreter','default');

    subplot(3,1,1)
    plot(t_WindowsCycles(:,1),I_WindowsCycles(:,1),'Color','b','LineWidth',2.0)
    set(gca,'fontSize',fontSize);
    title('Five Cycle of Luminesecence at Early Age','interpreter','latex');
    xlabel('Time (s)','interpreter','latex');
    ylabel(sprintf('%s(t) (V)', dataTypeSelected),'interpreter','latex');
    xlim([t_WindowsCycles(1,1) t_WindowsCycles(end,1)]);
    if normalize
        ylim([0 1.00]);
    end
    grid on
```

```
    subplot(3,1,2)
    plot(t_WindowsCycles(:,2),I_WindowsCycles(:,2),'Color','r','LineWidth',2.0)
    set(gca,'fontSize',fontSize);
    title('Five Cycle of Luminesecence at Intermediate Age','interpreter','latex');
    xlabel('Time (s)','interpreter','latex');
    ylabel(sprintf('%s(t) (V)', dataTypeSelected),'interpreter','latex');
    xlim([t_WindowsCycles(1,2) t_WindowsCycles(end,2)]);
    if normalize
        ylim([0 1.00]);
    end
    grid on

    subplot(3,1,3)
    plot(t_WindowsCycles(:,3),I_WindowsCycles(:,3),'Color',[0 0.5 0],'LineWidth',2.0)
    set(gca,'fontSize',fontSize);
    title('Five Cycle of Luminesecence at Near-failure Age','interpreter','latex');
    xlabel('Time (s)','interpreter','latex');
    ylabel(sprintf('%s(t) (V)', dataTypeSelected),'interpreter','latex');
    xlim([t_WindowsCycles(1,3) t_WindowsCycles(end,3)]);
    if normalize
        ylim([0 1.00]);
    end
    grid on

    CHECKMAKE('Figures');
    cd('Figures');
    %
saveas(RawDataPlot_I_windows,sprintf('Coup_%d_%s_Raw_Data_windows_cycles.fig',couponNumber(inde
x),dataTypeSelected));

saveas(RawDataPlot_I_windows_cycles,sprintf('Coup_%d_%s_Raw_Data_windows_cycles.png',couponNumb
er(index),dataTypeSelected));
    cd(currentFolder);

    %% Figure 2c: Plotting Raw Data I with sections highlighted in colors
    % Data Extraction for the plots
    [I_WindowsCyclesAmps(:,1),I_WindowsCyclesFrequency(:,1),I_WindowsCyclesPhase(:,1)] =
FFT(I_WindowsCycles(:,1),sampFreq);
    [I_WindowsCyclesAmps(:,2),I_WindowsCyclesFrequency(:,2),I_WindowsCyclesPhase(:,2)] =
FFT(I_WindowsCycles(:,2),sampFreq);
    [I_WindowsCyclesAmps(:,3),I_WindowsCyclesFrequency(:,3),I_WindowsCyclesPhase(:,3)] =
FFT(I_WindowsCycles(:,3),sampFreq);

    % Plotting
    DTFT_I_windows_cycles_Amp_Spectrum = figure('Name',sprintf('Coupon %d DTFT for Five Cycles
of %s(t)',couponNumber(index),dataTypeSelected));
    set(DTFT_I_windows_cycles_Amp_Spectrum,'Position',leftScreenFit);
    set(groot,'defaultLegendInterpreter','default');

    subplot(3,1,1)
    skippedPoints = 2;

plot(I_WindowsCyclesFrequency(skippedPoints+1:end,1),I_WindowsCyclesAmps(skippedPoints+1:end,1)
,'Color','b','LineWidth',2.0)
    set(gca,'fontSize',fontSize);
    DTFT_title = title('Early Age','interpreter','latex');
    DTFT_title.Position = [I_WindowsCyclesFrequency(end,1)/2
1.05*max(I_WindowsCyclesAmps(skippedPoints+1:end,1))];
    xlabel('Frequency (Hz)','interpreter','latex');
    ylabel(sprintf('%s(t) Amplitude (V)', dataTypeSelected),'interpreter','latex');
    grid on

    subplot(3,1,2)

plot(I_WindowsCyclesFrequency(skippedPoints+1:end,2),I_WindowsCyclesAmps(skippedPoints+1:end,2)
,'Color','r','LineWidth',2.0)
    set(gca,'fontSize',fontSize);
```

```
    DTFT_title = title('Intermediate Age','interpreter','latex');
    DTFT_title.Position = [I_WindowsCyclesFrequency(end,2)/2
max(I_WindowsCyclesAmps(skippedPoints+1:end,2))];
    xlabel('Frequency (Hz)','interpreter','latex');
    ylabel(sprintf('%s(t) Amplitude (V)', dataTypeSelected),'interpreter','latex');
    grid on

    subplot(3,1,3)

plot(I_WindowsCyclesFrequency(skippedPoints+1:end,3),I_WindowsCyclesAmps(skippedPoints+1:end,3)
,'Color',[0 0.5 0],'LineWidth',2.0)
    set(gca,'fontSize',fontSize);
    DTFT_title = title('Near-failure Age','interpreter','latex');
    DTFT_title.Position = [I_WindowsCyclesFrequency(end,3)/2
max(I_WindowsCyclesAmps(skippedPoints+1:end,3))];
    xlabel('Frequency (Hz)','interpreter','latex');
    ylabel(sprintf('%s(t) Amplitude (V)', dataTypeSelected),'interpreter','latex');
    grid on

    %  One title for all subplots
    subplotTitle = sgtitle('Luminiscence DTFT Amplitude Spectrums','interpreter','latex');
    subplotTitle.FontSize = 20;

    CHECKMAKE('Figures');
    cd('Figures');
    %
saveas(DTFT_I_windows_cycles_Amp_Spectrum,sprintf('Coup_%d_%s_DTFT_windows_cycles_Amp_Spectrum.
fig',couponNumber(index),dataTypeSelected));

saveas(DTFT_I_windows_cycles_Amp_Spectrum,sprintf('Coup_%d_%s_DTFT_windows_cycles_Amp_Spectrum.
png',couponNumber(index),dataTypeSelected));
    cd(currentFolder);

    %% Figure 3a: Plotting STFT 3-D Amplitude Spectrum
    % Data Extraction for the plots
    [I_WindowsAmps(:,1),I_WindowsFrequency(:,1),I_WindowsPhase(:,1)] =
FFT(I_Windows(:,1),sampFreq);
    [I_WindowsAmps(:,2),I_WindowsFrequency(:,2),I_WindowsPhase(:,2)] =
FFT(I_Windows(:,2),sampFreq);
    [I_WindowsAmps(:,3),I_WindowsFrequency(:,3),I_WindowsPhase(:,3)] =
FFT(I_Windows(:,3),sampFreq);

    % Remove amplitude due to shift
    skippedPoints = 10;
    I_WindowsAmps(1:skippedPoints,:) = 0;
    I_WindowsFrequency(1:skippedPoints,:) = 0;
    I_WindowsPhase(1:skippedPoints,:) = 0;

    % Plotting
    STFT_I_3D_Amp_Spectrum = figure('Name',sprintf('Coupon %d STFT 3-D Amplitude Spectrum of
%s(t)',couponNumber(index),dataTypeSelected));
    set(STFT_I_3D_Amp_Spectrum,'Position',leftScreenFit);
    set(groot,'defaultLegendInterpreter','default');


plot3(I_WindowsFrequency(:,1),100*ones(size(I_WindowsFrequency(:,1),1),1)*mean(t_Windows(:,1))/
t(end),I_WindowsAmps(:,1),'LineWidth',2.0,'Color','b');
    hold on;

plot3(I_WindowsFrequency(:,2),100*ones(size(I_WindowsFrequency(:,2),1),1)*mean(t_Windows(:,2))/
t(end),I_WindowsAmps(:,2),'LineWidth',2.0,'Color','r');

plot3(I_WindowsFrequency(:,3),100*ones(size(I_WindowsFrequency(:,3),1),1)*mean(t_Windows(:,3))/
t(end),I_WindowsAmps(:,3),'LineWidth',2.0,'Color',[0 0.5 0]);
    set(gca,'fontSize',fontSize);
    view(-45,25)
    title('STFT 3-D Amplitude Spectrum','interpreter','latex');
    xlabel('Frequency (Hz)','interpreter','latex');
```

```
    ylabel('Age (\% Failure)','interpreter','latex');
    zlabel(sprintf('%s(t) Amplitude (V)', dataTypeSelected),'interpreter','latex');
    yticks([0:10:100]);
    grid on
    legend('Early Age','Intermediate Age','Near-failure  Age','location','best')

    CHECKMAKE('Figures');
    cd('Figures');
    %
saveas(STFT_I_3D_cycles_Amp_Spectrum,sprintf('Coup_%d_%s_STFT_3D_Amp_Spectrum.fig',couponNumber
(index),dataTypeSelected));

saveas(STFT_I_3D_Amp_Spectrum,sprintf('Coup_%d_%s_STFT_3D_Amp_Spectrum.png',couponNumber(index)
,dataTypeSelected));
    cd(currentFolder);

    %% Figure 3a:Plotting of a hamming window
    % Data Extraction for the plots
    tHamming = [0:0.1:100]';
    dataHamming = hamming(max(size(tHamming)));

    % Plotting
    noBkgrdHammingWindow = figure('Name','Transparent background Hamming Window');
    set(noBkgrdHammingWindow,'Position',leftScreenFit);
    set(groot,'defaultLegendInterpreter','default');
    plot3(zeros(max(size(tHamming)),1),tHamming,dataHamming,'LineWidth',10.0)
    view(-45,25)
    set(gca,'xtick',[])
    set(gca,'ytick',[])
    set(gca,'ztick',[])
    axis off;
    set(gca,'color','none')
    CHECKMAKE('Figures');
    cd('Figures');
    % saveas(noBkgrdHammingWindow,'No_Background_Hamming_Window.fig');
    saveas(noBkgrdHammingWindow,'No_Background_Hamming_Window.png');
    cd(currentFolder);

    %% Figure 3b:Plotting STFT 3-D spectrogram
    % Data Extraction for the plots
    cyclicFrequency = [0:5:1000];

    % Plotting
    STFT_I_3D_Spectrogram = figure('Name',sprintf('Coupon %d STFT 3-D Spectrogram of
%s(t)',couponNumber(index),dataTypeSelected));
    set(STFT_I_3D_Spectrogram,'Position',leftScreenFit);
    set(groot,'defaultLegendInterpreter','default');

spectrogram(I,ceil(windowLength/100),ceil(windowLength/100*(overlapPercentage/100)),cyclicFrequ
ency,sampFreq);
    colormap hot
    view(35,60)
    shading interp
    colorbar off
    title('STFT 3-D Spectrogram');
    xlabel('Frequency (Hz)','interpreter','latex');
    ylabel('Age (\% Failure)','interpreter','latex');
    zlabel(sprintf('%s(t) Amplitude (V)', dataTypeSelected),'interpreter','latex');
    set(gca,'fontSize',fontSize+4);
    set(gca,'xtick',[])
    set(gca,'ytick',[])
    set(gca,'ztick',[])

    CHECKMAKE('Figures');
    cd('Figures');
    %
saveas(STFT_I_3D_Spectrogram,sprintf('Coup_%d_%s_STFT_3D_Spectrogram.fig',couponNumber(index),d
ataTypeSelected));
```

```
saveas(STFT_I_3D_Spectrogram,sprintf('Coup_%d_%s_STFT_3D_Spectrogram.png',couponNumber(index),d
ataTypeSelected));
    cd(currentFolder);

    %% Figure 3b:Plotting 2-D spectrogram
    % Data Extraction for the plots
    [A,F,T] =
spectrogram(I,ceil(windowLength/100),ceil(windowLength/100*(overlapPercentage/100)),cyclicFrequ
ency,sampFreq);

    % Plotting
    STFT_I_Spectrogram = figure('Name',sprintf('Coupon %d %s(t) STFT
Spectrogram',couponNumber(index),dataTypeSelected));
    set(STFT_I_Spectrogram,'Position',leftScreenFit);
    set(groot,'defaultLegendInterpreter','default');
    imagesc(F,[0 100],log10(abs(A))');
    colormap hot
    set(gca,'fontSize',fontSize);
    set(gca,'YDir','Normal')
    title(sprintf('%s(t) STFT Spectrogram',dataTypeSelected),'interpreter','latex');
    xlabel('Frequency (Hz)','interpreter','latex');
    ylabel('Age (\% Failure)','interpreter','latex');
    zlabel(sprintf('%s(t) Amplitude (V)', dataTypeSelected),'interpreter','latex');

    CHECKMAKE('Figures');
    cd('Figures');
%
saveas(STFT_I_Spectrogram,sprintf('Coup_%d_%s_STFT_Spectrogram.fig',couponNumber(index),dataTyp
eSelected));

saveas(STFT_I_Spectrogram,sprintf('Coup_%d_%s_STFT_Spectrogram.png',couponNumber(index),dataTyp
eSelected));
    cd(currentFolder);
end
```

```
%% Information
%{
 Author = Ahmed Mohamed
 Integrated Systems Lab at the Ohio State University
 Date = March 17th, 2020
 File name = MLPaperCode_2
 Premises = Extracting STFT Coefficients from saved .mat files, then
 implementing Artificial Neural Network for structural damage prognosis.
%}

%{
Files and functions needed:

Functions:
CHECKMAKE.m
NNCostFunction.m
fmincg.m
randInitializeWeights.m
sigmoid.m
sigmoidGradient.m

Files:
All data files named as following: cCouponNumber0FileNumber.mat
For instance: for CouponNumber = 25, FileNumber = 01 ---> fileName = c25001.mat

%}

%% Start
clear
clc
close all
tic                                                             % Starting a clock to
calculate time elapsed during algorithm run
currentFolder = cd;                                             % Saving main Folder
path
screenInformation = get(0,'screensize');                        % Obtaining screen
size
screenWidth = screenInformation(end-1);                         % Screen width
screenHeight = screenInformation(end);                          % Screen height
fontSize = 16;                                                  % Font size for all
plots
parentFolder = cd;                                              % Set directory to
Parent folder

%% Coefficients and Frequencies Extraction and Combining into a single .mat file
% Coupon and desired dataVector related constants
couponNumber = [25 27 38 39 40 41];                             % Coupon
Identification number
dataTypes = {'I','E','stress','strain'};                        % In case another
data type needs a similar analysis (let's say we want STFT for stress or strain vectors)
desiredDataTypes = {'I'};                                       % Desired data types
that the user wishes STFT data for
normalize = false;                                              % Deciding on using
normalized STFT coefficients or non-normalized ones

% Choosing STFT coefficients containing folder based on user's selection
if normalize
    CHECKMAKE('Normalized STFT Coefficients');
    NormSTFTCoeffientsFolder = cd('Normalized STFT Coefficients');
    for index = 1:length(couponNumber)
        for dataTypeSelector = 1:size(desiredDataTypes,2)
            data(index,dataTypeSelector) =
load(sprintf('Coup_%d_%s_Norm_STFT_Coefficients_&_Freqs',couponNumber(index),desiredDataT
ypes{dataTypeSelector}));
```

```
        end
    end
else
    CHECKMAKE('STFT Coefficients');
    STFTCoeffientsFolder = cd('STFT Coefficients');
    for index = 1:length(couponNumber)
        for dataTypeSelector = 1:size(desiredDataTypes,2)
            data(index,dataTypeSelector) =
load(sprintf('Coup_%d_%s_STFT_Coefficients_&_Freqs',couponNumber(index),desiredDataTypes{
dataTypeSelector}));
        end
    end
end
cd(parentFolder);

% Creating one data vector that has all coefficents and another vector that
... contains all frequencies
for idx = 1:size(data,1)
    I_CoefficientsCell{idx,1} = data(idx,1).STFTCoefficients;
    I_FrequenciesCell{idx,1} = data(idx,1).STFTFrequencies;
end

% Converting cells holding coefficients and frequencies to matrices
I_Coefficients = cell2mat(I_CoefficientsCell);
I_Frequencies = cell2mat(I_FrequenciesCell);

% Saving all coefficients and frequencies into 1 .mat file
if normalize
    cd('Norm STFT Coefficients')

save(sprintf('%s_Norm_STFT_Coefficients',dataTypes{dataTypeSelector}),'I_Coefficients');

save(sprintf('%s_Norm_STFT_Frequencies',dataTypes{dataTypeSelector}),'I_Frequencies');
else
    cd('STFT Coefficients')
    save(sprintf('%s_STFT_Coefficients',dataTypes{dataTypeSelector}),'I_Coefficients');
    save(sprintf('%s_STFT_Frequencies',dataTypes{dataTypeSelector}),'I_Frequencies');
end
cd(parentFolder);


%% ======================== Neural Network Start ========================
% Starting fresh
clearvars -except tic currentFolder screenInformation screenWidth ...
           screenHeight fontSize parentFolder couponNumber normalize...
           dataTypeSelector dataTypes STFTCoeffientsFolder Norm_STFT_Coefficients;

clc; close all;

%% Forward Propagation
% This whole section is  mostly redundant since the costFunction.m
... redefines all of these constants. Use this section to get familiarized
... to the different components of a neural network.

% Defining number of layers present in the architecture
... This is a design parameter defined by the user.
inputLayersNumber = 1;     % Number of input layers
outputLayersNumber = 1;    % Number of output layers
hiddenLayersNumber = 1;    % Number of hidden layers

% The total number of layers is simply the sum of input, output, and hidden
... layers
layersNumber = inputLayersNumber + outputLayersNumber + hiddenLayersNumber;
```

```
% If analysis is desired to be done on normalized data
if normalize
    cd('Norm STFT Coefficients')        % Open folder containing normalized data
    load(sprintf('%s_Norm_STFT_Coefficients',dataTypes{dataTypeSelector}));     % Load
normalized data to workspace
else
    cd('STFT Coefficients')            % Open folder containing non-normalized data
    load(sprintf('FFT_Peaks_Combined',dataTypes{dataTypeSelector}));          % Load non-
normalized data to workspace
end
cd(parentFolder);                       % Set current directory to parent folder

%% Implementing PCA on all Data
[EigenVectors,data_PCA,EigenValues,~,relativeSignificance] = pca(I_PEAKS);

% Selecting significant eigen vectors or features
chosenParameters = 4;                         % Only the first four features contribute
while the rest do not.

data_PCA = data_PCA(:,1:chosenParameters);

%% Neural Network
% Separating Training from testing data
trainingCoupons = 5;
testingCoupons = 1;
trainingExamplesPerCoupon = 100;

trainingData = data_PCA(1:trainingCoupons*trainingExamplesPerCoupon,:);
testingData = data_PCA(trainingCoupons*trainingExamplesPerCoupon+1:end,:);

% Input Layer specifications
inputLayerData = trainingData;             % The input is the light intensity STFT
Coefficients
inputLayerSize = size(inputLayerData,2);     % This number of input neurons should be 49
if
                                        ... all driving frequencies are considered.
trainingExamplesNumber = size(inputLayerData,1); % Number of training examples

% Output Layer specifications
numberOfDistinctAges = 100;              % The output size equal the number of available
different health states
                                        ... This number is also specified by the user for
a SDP problem.
                                        % For a digits recognition problem, the number of
output layer
                                        ... neurons is equal to the number of distinct
characters to be recognized.
outputLayerData = linspace(1,100,numberOfDistinctAges)'; % Output layer data
outputLayerSize = size(outputLayerData,1); % Output Layer size

% In case the data are aranged by age
% Defining the reference output layer
referenceOutputLayer = zeros(outputLayerSize,trainingExamplesNumber);
for indx = 1:size(referenceOutputLayer,1)
    for idx = 1:size(referenceOutputLayer,2)
        if ceil(idx/max(size(couponNumber))) == indx
            referenceOutputLayer(indx,idx) = 1;
        end
    end
end

% Since coupon's data are stacked up on each others
```

```
% Defining the reference output layer
% referenceOutputLayer = [];
% for idx = 1:max(size(couponNumber))
%     referenceOutputLayer = cat(2,referenceOutputLayer, eye(numberOfDistinctAges));
% end

% We defined the output layer to be 0%,10%,20%, ... 100% failure

% Hidden Layers
hiddenLayersSize = 8;   % Number of neurons in the hidden layer is recommended to be less
than
                                        ... or equal to the number of neurons in the
input layer. (Literature)
hiddenLayers{1} = rand*ones(hiddenLayersSize,1); % Randomly initializing values to
populate the hidden layer neurons

% Note: this section is used to visualize the different components of an
... artificial neural network.



%% Now Forward propagation
% Note: Again this section is used to visualize the different components of
... an artificial neural network, which may seem redundant

% Initializing Activation Neurons of each layer (We denote the letter 'A'
... as it stands for activation)

% All the layers are saved into a single cell array called A, which has a
... flexible size depending on the number of layers specified by the user.
A = cell(layersNumber,1);
for idx = 1:size(A,1)
    if idx == 1
        A{idx,1} = inputLayerData(1,:)'; % An input layer is a matrix of the
                                    ... following dimensions:
                                    ... (TrainingExamples * InputLayerNeurons)
    elseif idx == size(A,1)
    A{idx,1} = outputLayerData; % An output layer is usually a column vector
    else
        A{idx,1} = hiddenLayers{idx-1}; % Can take more than 1 hidden layer
    end
end



% Initializing weights. Weights are usually donated the greek letter theta
Theta = cell(layersNumber-1,1);
% Initiating an empty cell array with the correct dimensions
... (Theta's size is usually 1 cell less than the size of A)
% If three layers are present, then two weights cells are needed to connect
... the three layers.
% If four layers are present, then three weights cells are needed to
... connect the four layers, and so on.

for idx = 1:size(Theta,1)
    if idx == 1
        Theta{idx,1} = randInitializeWeights(inputLayerSize,hiddenLayersSize);
    elseif idx == size(Theta,1)
        Theta{idx,1} = randInitializeWeights(hiddenLayersSize,outputLayerSize);
    else
        Theta{idx,1} = randInitializeWeights(hiddenLayersSize(idx-
1),hiddenLayersSize(idx));
    end
end
```

```
% Creating a vector of weight parameters (Unrolling the weights matrices)
ThetaVector = [];
for idx = 1:size(Theta,1)
    ThetaVector = [ThetaVector; Theta{idx,1}(:)];
end

cd(parentFolder);   % Set current directory to parent folder

%% Computing Cost Function  (Back Propagation)
% Weight regularization parameter (we set this to 0 here).
lambda = 0.01;
% Regularization is a technique used to avoid overfitting of weights during
... the optimization process.
% It is observed that the larger lambda, the quicker the optimization
... parameter converges as it tries to avoid overfitting the decision
... boundary between different data clusters or data categories

LayersDimensions = [inputLayerSize hiddenLayersSize outputLayerSize]';
% Computing the cost associated with the randomly assigned weights
% connecting the different layers of an Artificial Neural Network.
[J,Gradient] = NNCostFunction(ThetaVector, LayersDimensions, inputLayerData,
referenceOutputLayer, lambda);



% Specifing the optimization stop condition to be based on maximum
... iterations, hence 'MaxIter'. The second argument is the desired number
... of iterations the user wants the optimization function to execute.
options = optimset('MaxIter', 5000); % Stop-condition is set to iterations number
                                     % Maximum  umber of iterations = 5000;
% Create "short-hand" notation for the cost function to be minimized to be
... inputted to the optimization function (fmincg.m)
costFunction = @(p) NNCostFunction(p, LayersDimensions, inputLayerData,
referenceOutputLayer, lambda);

% Now, costFunction is a function that takes in only one argument (the
% neural network parameters)

% Below is the optimization function fmincg.m. This function is out-sourced
% This function optimizes using multivariate regression in an efficient way
... compared to fminunc.m (MATLAB's standard multivariate regression
... optimization function)
[ThetaVectorFinal] = fmincg(costFunction, ThetaVector, options);

% Obtaining each Theta matrix back from the column vector of weights
... ThetaVectorFinal
ThetaFinal{1} = reshape(ThetaVectorFinal(1:hiddenLayersSize* (inputLayerSize + 1)), ...
                hiddenLayersSize, (inputLayerSize + 1));

ThetaFinal{2} = reshape(ThetaVectorFinal((1 + (hiddenLayersSize * (inputLayerSize +
1))):end), ...
                outputLayerSize, (hiddenLayersSize + 1));



%% Testing 5-to-1 split
% Testing Phase on training data
for indx = 1:trainingCoupons
    for idx = 1:trainingExamplesPerCoupon

     ActualAge(idx,indx) = idx;
     testSection = [1 trainingData(idx,:)]';
        Layer1 = sum(ThetaFinal{1}'.*testSection,1);
```

```
        Layer1 = [1 sigmoid(Layer1)]';
        Layer2 = sum(ThetaFinal{2}'.*Layer1,1);
        Layer2 = sigmoid(Layer2)';

        [MaxIndex,chosenIndex] = max(Layer2);
        AgeFound(idx,indx) = outputLayerData(chosenIndex);

    end
    ageError(:,indx) = AgeFound(:,indx) - ActualAge(:,indx);
    nonZeroError = ageError(ageError(:,indx)~=0);
    PercentageError(:,indx) =
100*(length(nonZeroError)/(size(ageError,1)*size(ageError,2)));
end
ageError = abs(ageError/trainingExamplesPerCoupon)*100;
Accuracy = 100-ageError;

% Testing Phase on testing data
AverageError = mean(PercentageError);

for indx = 1:testingCoupons
    for idx = 1:trainingExamplesPerCoupon

     ActualAge_test(idx,indx) = idx;
     testSection = [1 testingData(idx,:)]';
        Layer1 = sum(ThetaFinal{1}'.*testSection,1);
        Layer1 = [1 sigmoid(Layer1)]';
        Layer2 = sum(ThetaFinal{2}'.*Layer1,1);
        Layer2 = sigmoid(Layer2)';

        [MaxIndex,chosenIndex] = max(Layer2);
        AgeFound_test(idx,indx) = outputLayerData(chosenIndex);

    end
    ageError_test(:,indx) = AgeFound_test(:,indx) - ActualAge_test(:,indx);
    nonZeroError_test = ageError_test(ageError_test(:,indx)~=0);
    PercentageError_test(:,indx) =
100*(length(nonZeroError_test)/(size(ageError_test,1)*size(ageError_test,2)));
end
ageError_test = abs(ageError_test/trainingExamplesPerCoupon)*100;
Accuracy_test = 100-ageError_test;

AverageError_test = mean(PercentageError_test);
```